BACHELORTHESIS

Moderner Workflow zur Erstellung von Frontend-Templates mit Anbindung an Content Management Systeme.

Konzeption, Implementierung und Realisierung eines Prototypen.

von	Lena Basedow
Betreuer	Prof. Dr. Roland Riempp (1. Betreuer) Yannick Herzog, B. Sc. (2. Betreuer)
Hochschule	Hochschule Offenburg Medien und Informationswesen
Semester	Wintersemester 2015/16

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Offenburg, den

Lena Basedow

Inhalt

1 Einleitung

„Persönlichkeiten werden nicht durch schöne Reden geformt, sondern durch Arbeit und eigene Leistung."

Albert Einstein

1.1 Das Thema

Der Fokus dieser Thesis liegt auf der Erarbeitung eines modernen Workflows zur Erstellung von Frontend-Templates. Diese Arbeitsweise soll später im Alltag als Webentwicklerin Anwendung finden.

Dieser Fokus wurde gewählt, da es eine allgemeine, aber auch sinnvolle Vorbereitung auf die Arbeit in der Webentwicklung darstellt. Egal ob als Freiberuflerin oder als Angestellte in einer Agentur. Jede Agentur sowie manche Kunden haben ihre eigenen Anforderungen welches Content Management System (CMS) verwendet werden soll. Um effektiv und bis zu einem bestimmten Grad flexibel zu bleiben, wird der Workflow für Frontend-Templates völlig unabhängig zu einem CMS gelassen. Egal welches CMS der Auftraggeber für sich beansprucht, die Arbeitsschritte bis zum fertigen, technisch umgesetzten Design werden die Gleichen sein. Erst im darauffolgenden Schritt wird das Frontend-Template mit dem Content Management System verbunden.

Diese Arbeit bringt neben dem Workflow noch weitere Ergebnisse hervor. Zum Einen ein flexibles Template, das über das Farbkonzept und einem umfangreichen Modulkatalog für verschiedene Kunden leicht angepasst werden kann. Des Weiteren wird es eine beispielhafte Integration für einen fiktiven Kunden in ein CMS geben. Um die Vorteile der Ergebnisse noch zu verdeutlichen, zwei Szenarien:

Es geht ein Auftrag von einem Kunden A ein, der Relaunch seines Webauftrittes. Dem Kunden wird das Frontend-Template vorgestellt, welches ihm gefällt. Nun wird es direkt an das von ihm bisher genutzte CMS gebunden.

Ein Kunde B möchte eine neue Webseite, allerdings sagt ihm keines der bisher bestehenden Frontend-Templates zu. Also wird am Anfang des Workflows begonnen und ein neues Design erstellt.

Bei Kunde A kann bereits beim Schritt „Anbindung an ein CMS“ begonnen werden (Kapitel 7). Bei Kunde B wird am Anfang, bei der Erstellung des Designs (Kapitel 5), begonnen.

1.2 Der Ablauf der Thesis

Die Thesis behandelt einen theoretischen und einen praktischen Teil. Der praktische Teil bildet den Workflow zur Erstellung von Frontend-Templates mit anschließender Anbindung an ein Content Management System. Die „Abbildung 1.1.: Ablauf des Workflows“ auf Seite 13 veranschaulicht den Ablauf des Workflows.

Der Theoretische Teil

Zunächst werden aktuelle Trends, Design und Markt, rund um das Thema Templates für Content Management Systemen untersucht. Im nächsten Kapitel werden die im praktischen Teil angewandten Techniken zur Erstellung von Frontend-Templates vorgestellt. Darauffolgend wird auf die Beschreibung und den Vergleich von Content Management Systemen eingegangen.

Der Workflow (praktischer Teil)

Der Workflow beginnt mit der Erarbeitung des Konzeptes des Templates (Kapitel 5). Es wird festgelegt, welchen Aufbau, welche Module und welches Design das Template bekommt und anschließend mit einem Wireframing-Tool und Photoshop als PDF-Dateien realisiert. Da der Fokus auf der Entwicklung liegt, fällt der Umfang des Designs eher schlicht aus.

Der nächste Schritt ist die technische Umsetzung des im vorherigen Kapitels erstellten Designs, u.a. mit HTML, CSS und JavaScript (Kapitel 6). Das Ergebnis wird eine Demo-Seite sein, in der alle Möglichkeiten des Templates zu sehen sind. Bis hierhin besteht keine Verbindung zu einem Content Management System.

Als Nächstes kommt die Integration in ein CMS (Kapitel 7). Hierzu simuliert dieses Kapitel den Auftrag eines fiktiven Kunden, in dem Fall eine Rechtsanwaltskanzlei, die einen Relaunch des aktuellen Internetauftrittes möchte. Anhand der Demo-Seite aus dem vorherigen Schritt (Kapitel 6) entscheidet er welche Module später in seiner Webseite integriert werden sollen. Zusätzlich bringt der Kunde sein eigenes Farbkonzept und die Vorgabe ein bestimmtes Content Management System zu verwenden, mit.

Der Abschluss

Zum Abschluss wird der Workflow noch einmal zusammengefasst und ein Fazit über den Verlauf der Erarbeitung gezogen.

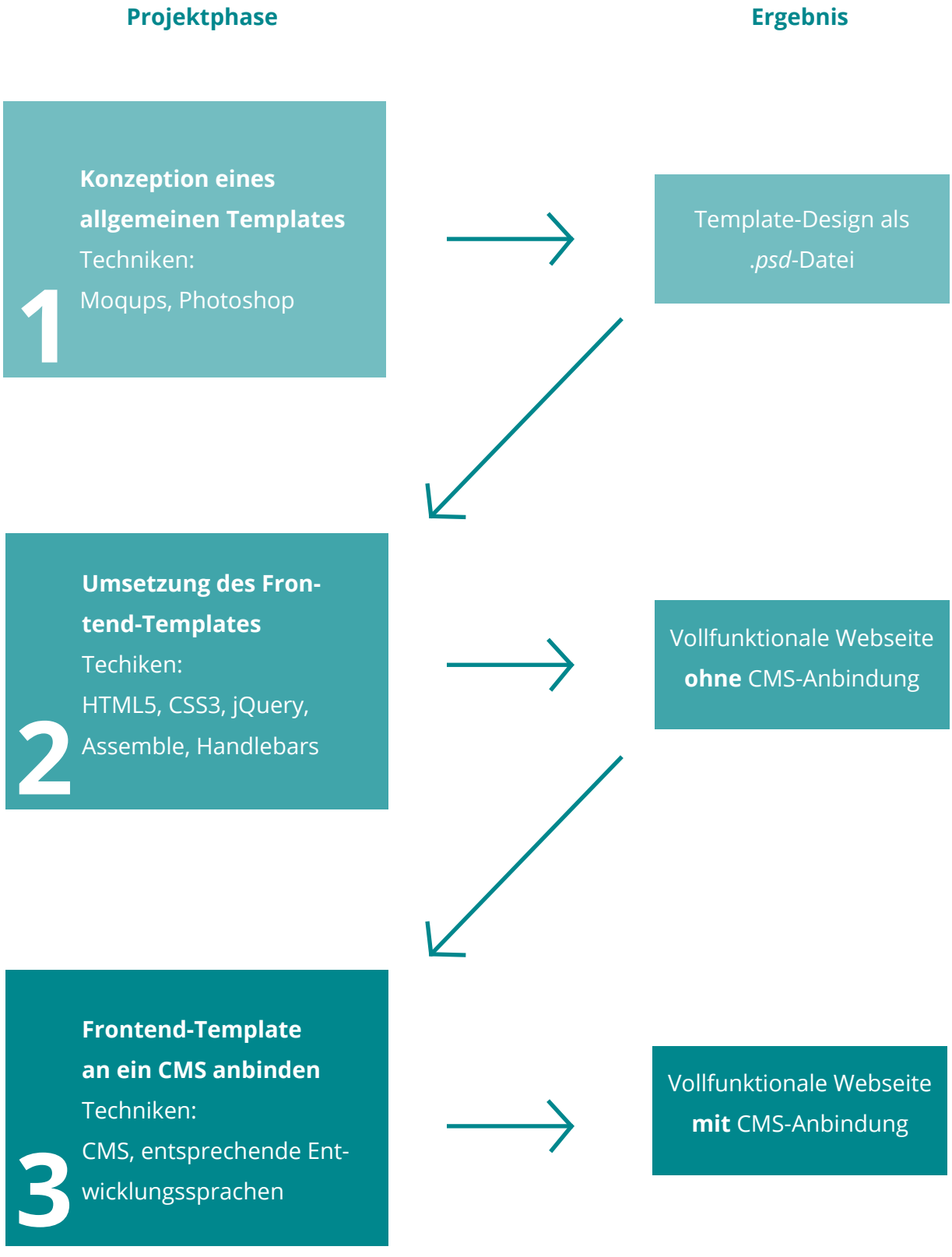


Abbildung 1.1: Ablauf des Workflows (praktischer Teil)

2 Templates

2.1 Definition

Als Template wird eine Vorlage bezeichnet, nach dessen Vorbild Kopien mit unterschiedlichen Inhalten und kleinen Anpassungen angefertigt werden können. Im Bereich der Webentwicklung gilt ein Template als Designvorlage und wird auch als „Theme“ bezeichnet. Es bestimmt die Anordnung von Elementen, Schrifteinstellungen, Farben usw. Alles rund um das Aussehen einer Webseite. Wird ein Template dynamisiert, also mit einem Content Management System verbunden und werden Inhalte (Bilder, Texte, etc.) eingepflegt, entsteht eine individuelle Internetseite. Mit Hilfe eines CMS kann auch jemand ohne Kenntnisse von Webtechnologien wie HTML die Inhalte einer Webseite bearbeiten.

2.2 Der Template-Markt

Als Interessent einer neuen Webseite gibt es zwei Möglichkeiten. Zum Einen die Option sich ein fertiges Template für ein bestimmtes CMS online zu kaufen oder eine Agentur oder einen Webentwickler zu beauftragen, um ein personalisiertes Template erstellen zu lassen. Entscheidet er sich für die erste Option, steht der Interessent vor einer großen Auswahl im Internet. Es finden sich zahlreiche Anbieter, die sich allein auf den Vertrieb von Templates spezialisiert haben, meist beschränkt auf Templates für ein oder wenige Content Management Systeme, wobei Wordpress ungeschlagener Spitzenreiter ist, wenn es um die verfügbaren Themes geht. Allein das Angebot des Anbieters themeforest verfügt insgesamt über 21.000 Vorlagen, wovon knapp 6000 für Wordpress sind (*themeforest.net,* 2015).

Dazu kommen über 46.000 Designs auf *templatemonster.com* (nicht nur für Wordpress). Wer sich von solch einer Auswahl erschlagen fühlt, schaut sich die im Vergleich bescheidene Anzahl von Templates von elegantthemes (87 Vorlagen) (*elegantthemes.com,* 2015), wpzoom (über 50) (*wpzoom.com,* 2015) oder WooThemes (über 100) (*awesemthemes.de,* 2015) an. Ebenso findet man eine Reihe an Themes auf den offiziellen Webseiten des jeweiligen Content Management Systems.

Kommt es nun zum Bezahlen, gibt es ebenfalls mehrere Möglichkeiten. Die klassische Variante ist der Einzelpreis. Die Preisspanne liegt ungefähr zwischen 0 $ (vgl. Brinkmann, 2015) bis 149 $ (*awesemthemes.de,* 2015), wobei verschiedene Optionen enthalten oder eben nicht enthalten sind. Beim Anbieter elmastudio sind im Kauf u.a. eine unbegrenzte Theme-Nutzung, ein Jahr Updates und ein Support über ein Forum sowie die Dokumentation im Preis enthalten (*elmastudio.de,* 2015).

Neben dem Einzelpreis gibt es die Abovariante, bei der man in regelmäßigen Abständen Beiträge zahlt und sich so mindestens Zugang zu allen angebotenen Themes verschafft. Elegantthemes unterscheidet drei Abonements. Variante 1 „Personal“ beinhaltet für 69 $ pro Jahr Zugang zu allen Themes, Updates, technischem „Premiumsupport“ im Forum und die Nutzung der Themes auf einer unbegrenzten Anzahl von Seiten. Aufgepeppt wird dies durch Variante 2 „Developer“ für 89 $, welches noch zusätzlichen Zugang zu allen aktuellen und während der Mitgliedschaft entwickelten Plugins sowie den Photoshop-Dateien der Designs verschafft. Die dritte Variante entfernt sich etwas von den Abonements, indem eine lebenslange Mitgliedschaft mit allen Vorzügen von

„Developer“ mit einer Einmalzahlung von 249 $ erworben werden kann (*elegantthemes.com,* 2015). Zum Vergleich: WPZoom verkauft alle Features der Variante „Developer“ von elegantthemes für einmalig 149$ (*wpzoom.com,* 2015).

Im Gegenzug dazu ist die Preisspanne von individuell gefertigten Webseiten immens groß. Nach eigenen Erfahrungen beginnend im niedrigen dreistelligen Bereich für sehr kleine Seiten ohne Unterseiten, bis zum 6-stelligen Bereich für komplexe, internationale Webauftritte.

2.3 Aktuelle Trends

Wie auch in der Mode, Kunst und Design entwickeln sich die Looks von Webseiten stets weiter. Die aktuellen Trends und Stile, die sich durch eine komplette Seite ziehen, sind nicht nur von rein optischen Geschmäckern beeinflusst sondern auch von den Entwicklungen der Technik (vgl. Hahn, 2015, S. 673). Im Folgenden werden einige aktuelle Webdesigntrends näher vorgestellt.

2.3.1 Responsive Webdesign

Im Jahr 2007 nahm alles seinen Anfang. Apple entwickelte das erste IPhone und brachte somit den Stein für mobiles Surfen ins Rollen. Bis dato war es üblich die Internetseiten auf eine feste Breite von 960px zu beschränken (Abbildung 2.1). Seit dem entwickeln sich die internetfähigen Geräte stets weiter. So war es zu Anfang nur der standartmäßige Computer für den Webseiten erstellt wurden. Doch heute ist bei Weitem nicht mehr nur dieses Gerät entscheidend. Das Web hat so einige weitere Devices erobert wie Smart-

phones, Tablets und SmartTVs (Abbildung 2.2). Und damit ist die Entwicklung noch lange nicht abgeschlossen, durch das Web 3.0 zählen bald auch Autos, Kühlschränke und andere Haushaltsgegenstände zu dem Testarsenal der Webentwickler (Abbildung 2.3. Die Anforderung: die Webseiten müssen auf allen Größen benutzerfreundlich bleiben und gut aussehen (vgl. Hahn, 2015, S.136).

Abbildung 2.1:
Das Web damals
(*bradfrost.com*)

Abbildung 2.2:
Das Web heute
(*bradfrost.com*)

Abbildung 2.3:
Das Web morgen ???
(*bradfrost.com*)

Und spätestens seit Google im April 2015 bei mobilen Suchanfragen für mobile Geräte optimierte Seiten bevorzugt, darf Responisve Design nicht mehr ignoriert werden. Für Template-Entwickler ist dieser Trend schon jetzt ein fester Bestandteil ihres Workflows; unter den angebotenen Themes auf dem Markt entdeckt man kaum mehr eines, das nicht responsiv ist.

CSS Media Queries

CSS3 Media-Queries legen in einer CSS-Datei fest, welche Styles bei welcher Bildschirmbreite zu Einsatz kommen.

Aber was bedeutet „Responsive Web Design" eigentlich genau? Der Online-Autor Venkatesh beschreibt es auf einer Infografik (Abbildung 2.4) als „Ansatz, der vorschlägt, dass Design und Entwicklung auf das Verhalten und die Umgebung bezüglich Bildschirmgröße, Plattform und Bildschirmorientierung des Nutzers reagieren sollten" (Venkatesh, 2012). Des Weiteren wird in der Grafik gezeigt, dass sich Responsive Web Design aus einer URL, Content, einem Code und CSS3 Media Queries zusammensetzt.

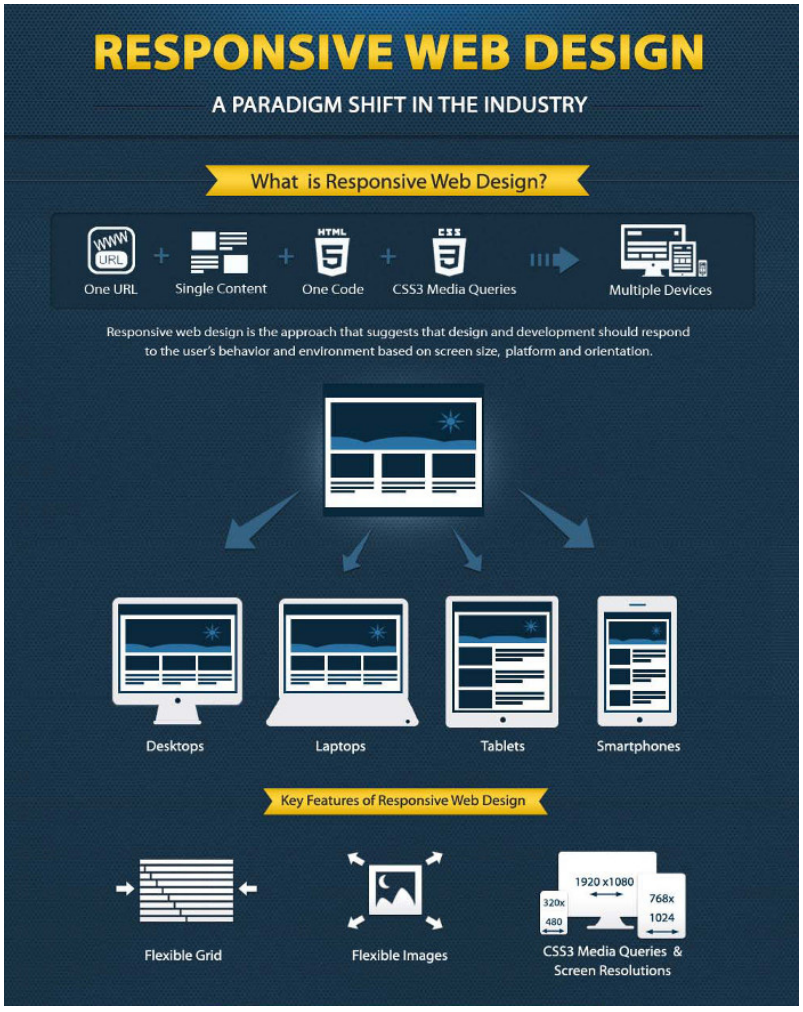


Abbildung 2.4: Ausschnitt Infografik „Responsive Web Design" (*dotcominfoway.com*)

Weitere Merkmale sind Bildgrößen, die sich prozentual zur Bildschirmgröße verändern, ein flexibles Gridsystem und Media Queries mit einer begrenzten Anzahl an Bildschirmbreiten.

2.3.2 Flat Design bis Material Design

Vor einigen Jahren entwickelte Microsoft mit Windows 8 einen neuen Design-Stil, der den bis dahin langanhaltenden Skeuomorphismus von Apple ablöste: Flat Design.

Skeuomorphismus
Dieser Stil bemüht sich Oberflächen möglichst realistisch und intuitiv darzustellen.

Flat Design ist das Gegenteil von Skeuomorphismus. In diesem Fall werden Oberflächenstrukturen, Haptik, Schattierungen, Animationen und Dreidimensionalität durch „kräftige Farben, markante Typografie, die auf serifenlose schlanke Schriftarten setzt, einfarbige Icons und ein klares Gestaltungsraster“ (Hahn, 2015, S. 678) ersetzt.

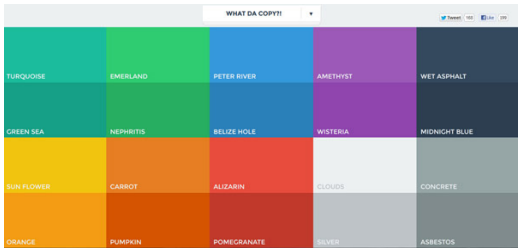


Abbildung 2.5: Auswahl an Farben für Flat Design (*flatuicolors.com*)

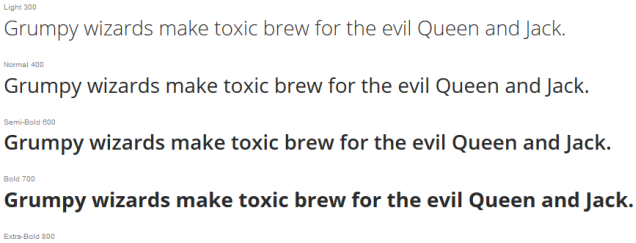


Abbildung 2.6: Google Font „Open Sans“ (*google.com/fonts*)

Das eher minimalistische Design, welches vorwiegend durch CSS umgesetzt werden kann, ermöglicht natürlich schnellere Ladezeiten als im Skeuomorphismus, da auf aufwändige Grafiken verzichtet wird. Dies schont, zur Freude der Nutzer, das Datenvolumen. Ebenso fördert es die Übersichtlichkeit von Webseiten. Als Nachteil kann der geringe Wiedererkennungswert von Benutzeroberflächen gesehen werden (vgl. *t3n.de/tag*). Dazu besteht bei starker Reduzierung die Gefahr, dass Benutzer klickbare Elemente nicht auf Anhieb von nicht klickbaren Elementen unterscheiden können, da beide eine einfarbige Fläche haben (vgl. Hahn, 2015, S. 679).

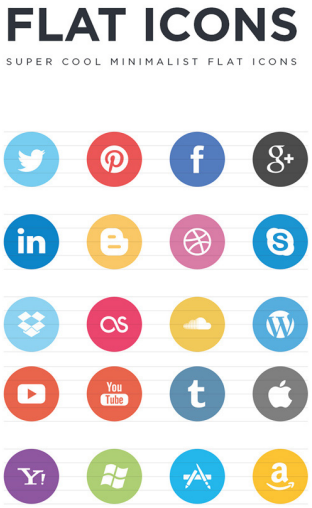


Abbildung 2.7: Mögliche Icons für Flat Design (*behance.net*)

Neben Apple und Microsoft mischte sich Mitte 2014 auch Google unter die Webdesign-Trendsetter und präsentierte ihren Styleguide zu „Material Design“. Auf der offiziellen Styleguide Webseite heißt es laut Google „We challenged ourselves to create a visual language for our users that synthesizes the classic principles of good design with the innovation and possibility of technology and science. This is material design.“ (*google.com/design,* 2015). Demnach baut Material Design auf die Kombination von „gutem Design” und den Innovationen und Möglichkeiten von Technologie und Wissenschaft. Als neutrale Person beschreibt Martin Hahn den Stil als dem Flat Design sehr ähnlich, allerdings noch um die z-Achse erweitert (vgl. Hahn, 2015, S. 687). Dabei sorgt die z-Achse nur für die Stapelung der Elemente, nicht für die Perspektive (*google.com/design*). Alle Elemente haben die Eigenschaft 1 px hoch zu sein und werden auch häufig als „Papier-Elemente“ bezeichnet (Abbildung 2.9).

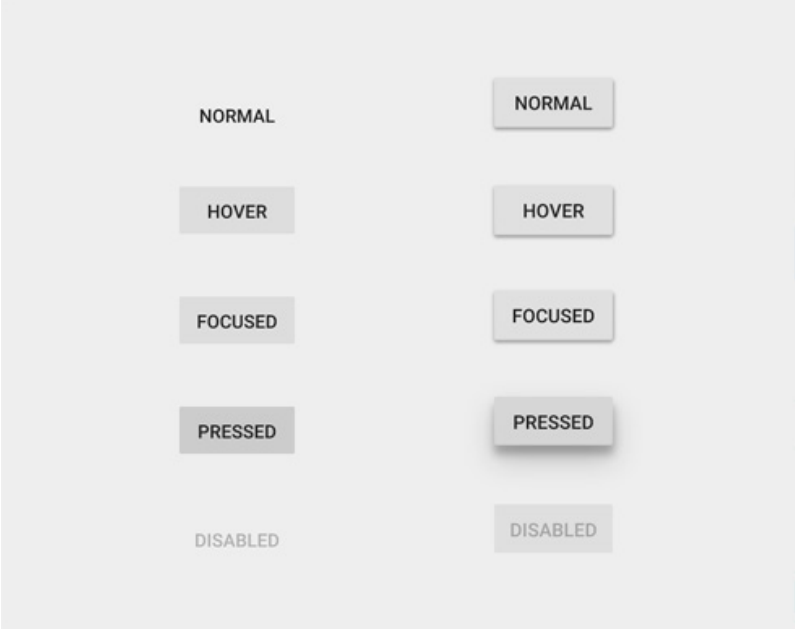


Abbildung 2.8: Direkter Vergleich von Buttons im Flat Design (links) mit Buttons im Material Design (rechts) (*t3premium.de/blog*)

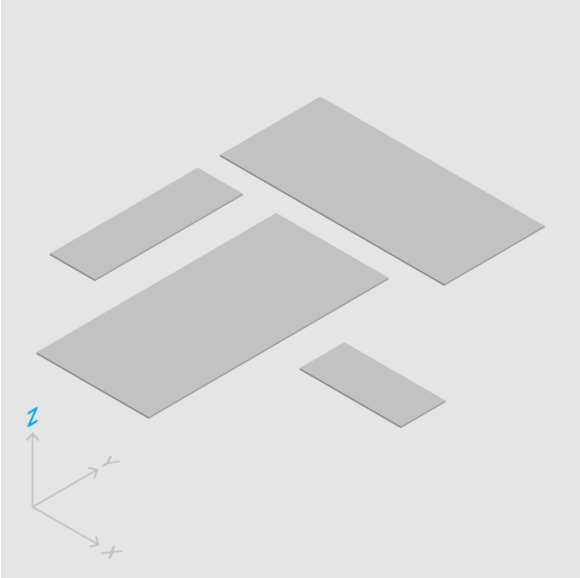


Abbildung 2.9: Darstellung von Elementen im Material Design mit z-Achse (*google.com/design*)

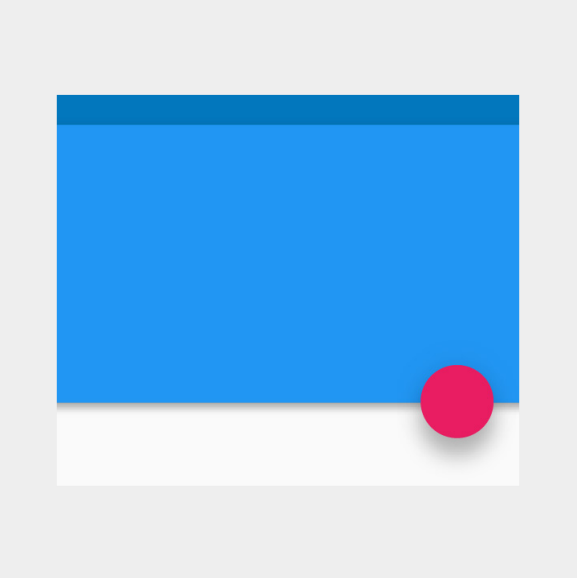


Abbildung 2.10: Elementen im Material Design übereinander gestapelt (*google.com/design*)

Damit der User die Anordnung auf unterschiedlichen Ebenen bemerkt, werden Schatten eingesetzt. Je weicher und weitlaufender der Schatten ist, desto höher liegt das Element im Gegensatz zu den anderen (Abbildung 2.10).

Für die optimierte Umsetzung von Material Design auf Webseiten hat Google Mitte dieses Jahres die Bibliothek „Material Design Lite“ veröffentlicht. Sie besteht aus Komponenten und Templates aus HTML, CSS und JavaScript und lässt sich mit nur wenigen Abhängigkeiten und einer geringen Größe von nur 27 Kilobyte leicht integrieren (vgl. Bonset, 2015).

2.3.3 One Pager

Webseiten, die nur aus einer einzigen Seite bestehen, werden One Pager genannt. Hierbei fällt das lästige Laden von neuen Seiten weg. Empfehlenswert ist dieser Trend für Webauftritte mit überschaubarem Content. Alle Informationen stehen in Abschnitte unterteilt untereinander. Die einzelnen Abschnitte lassen sich meist durch eine Navigation direkt ansteuern. Durch einen im besten Fall fließenden Scroll-Effekt erscheint dann der gewünschte Bereich im Sichtfeld des Users.

Neben Anwendung für den Unternehmens- oder Personenauftritt, wird diese Art des Webdesigns auch häufig für „Storytelling" verwendet. Beim Storytelling wird dem User der Content durch Texte, Grafiken, Bilder, Videos, Userinteraktionen und natürlich durch jede Menge Scrollen nähergebracht (vgl. Bauer, 2016). Auf der Seite der Space Needle (*spaceneedle.com*) in Seattle darf der User die Fahrt mit dem Lift bis nach oben und noch darüber hinaus durch eigenständiges Scrollen selbst bestimmen. Während der Fahrt erscheinen Kreuze, die durch Anklicken einen kurzen Infotext zeigen. Auf der Plattform der Space Needle angekommen, bekommt der User die Möglichkeit eine 360°-Sicht über die Stadt Seattle zu genießen und sie gleichzeitig durch Infotexte zu erkunden. Als weiteres Beispiel dient die preisgekrönte OnePageSite *flatvsrealism.com,* die über Scrollen den „Kampf" von Flat Design gegen den Skeuomorphismus beschreibt (Abbildungen 2.11 und 2.12).

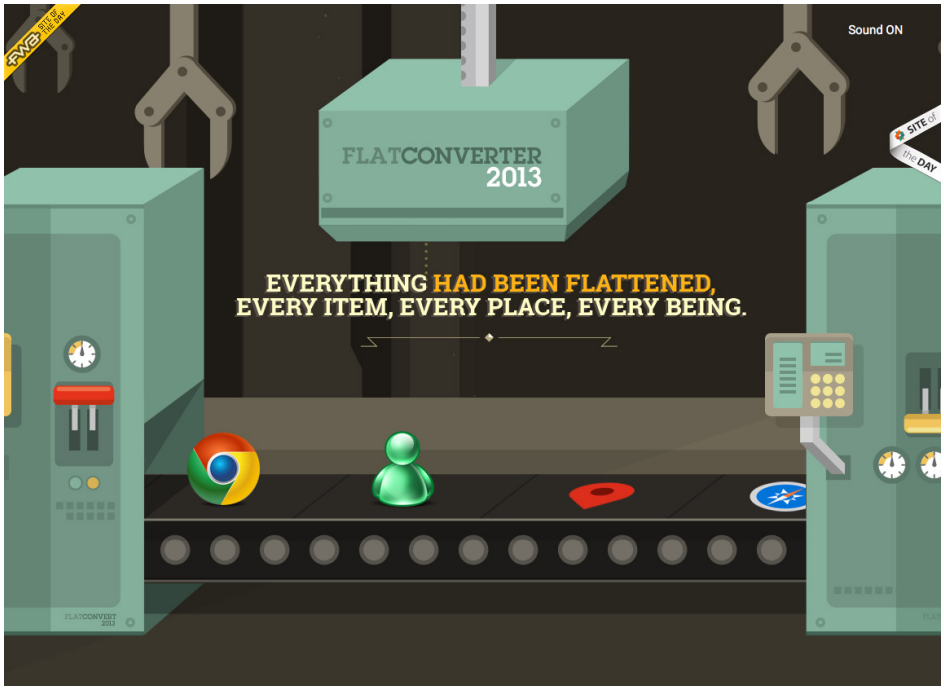


Abbildung 2.11: Die Geschichte des Wandels vom Skeuomorphismus zum Flat Design als One-Pager. (*flatvsrealism.com*)

Abbildung 2.12: Die Geschichte endet mit einem Spiel in dem der User entweder Flat Design oder Skeuomorphismus vertritt. (*flatvsrealism.com*)

2.4 Vor- und Nachteile: Unternehmen

Bevor ein Unternehmen den nächsten Online-Shop für Webtemplates aufsucht, sollte es die Vor- und Nachteile eines fertigen Templates abwägen (*klickkomplizen.de*, 2010).

Vorteile

1. Mit wenig Zeitaufwand bekommt das Unternehmen ein komplett fertiges, funktionales Design für das Wunsch-CMS und braucht nur noch die Inhalte einpflegen.
2. Im Vergleich zu einem individuell entwickelten Design, ist der Kauf eines fertigen Templates kostengünstiger.

Nachteile

1. Bei einem Online erworbenem Template besteht kein direkter Ansprechpartner oder nur für eine gewisse Zeitspanne. Sollte es Fragen zum Einrichten oder Verwalten der Webseite mit dem Template durch das CMS nach Ablauf der Supportzeit geben, muss das Problem über Internetrecherche angegangen werden.
2. Bei einem vorgefertigten Template besteht die Gefahr, dass auch weitere Unternehmen, im schlimmsten Fall direkte Konkurrenten, dieses für Ihre Webseite verwenden.
3. Google ändert regelmäßig die Algorithmen für ihre Suchmaschine. Ältere Templates, für die der Anbieter keine Updates mehr veröffentlicht, könnten nicht mehr den aktuellen SEO-Richtlinien entsprechen und bei Suchanfragen weniger berücksichtigt werden.
4. Steuert das Unternehmen einen internationalen Markt an, sollte das Template in der Lage sein, sich anzupassen. Zum Beispiel an die von rechts nach links geschriebenen Texte aus asiatischen Ländern.

5. Ein fertiges Template hat seine Grenzen und lässt sich nur bis zu einem bestimmten Grad den eigenen Vorstellungen anpassen.
6. Es bedarf einen gewissen Zeitaufwand, das Template und den Content einzupflegen, die der Webseitenbetreiber aufbringen muss.

Am größten wiegen allerdings die Punkte Budget und geplanter Content. Ist das Budget klein wird ein fertiges Template ausreichen müssen. Soll es nur eine kleine Webseite werden wie eine Art Online-Visitenkarte, reicht ebenfalls ein fertiges Template (*klickkomplizen.de*, 2010).

2.5 Vor- und Nachteile: Entwickler

Vorteile

1. Ein Nachteil für Template-Nachfrager wird zum Vorteil von Template-Entwicklern: die Wiederverwendbarkeit.
2. Ist das Template für keinen bestimmten Kunden, hat der Entwickler vollkommene Freiheit in Bezug auf Design, Umfang und verwendete Techniken (z.B. neue CSS3-Techniken, die nicht unbedingt von jedem älteren Browser unterstützt werden).

Nachteile

1. Je nach CMS ist das bestehende Angebot von Templates enorm und die Wahrscheinlichkeit, dass ein Interessent sich das eigene Template anschaut, geschweige denn sich dafür entscheidet, ist geringt.
2. Viele Käufer sehen nicht den Arbeitsaufwand der hinter einem Template steckt und sind daher meist nicht bereit angemessen dafür zu bezahlen.

3. Der Arbeitsaufwand für ein fertiges Template ist erst, je nach Preis, nach x verkauften Kopien entschädigt.

Rentabler für einen Entwickler sind natürlich individuelle Kundenaufträge. Sollte einmal eine Auftragsflaute bestehen, ist es zu empfehlen Themes über Online-Plattformen anzubieten, die sich ohne Eigenaufwand selbst anbieten und verkaufen.

3 Techniken zur Erstellung von Frontend-Templates

Die Webentwicklung umfasst zahlreiche Techniken und Hilfsmittel für die Umsetzung von Projekten. Dieses Kapitel soll eine Einführung in die Template-Entwicklung geben und die im praktischen Teil angewandten Tools und Methoden genauer erläutern.

3.1 Definition Frontend

Die Webentwicklung lässt sich in zwei Bereiche unterteilen. Zum Ersten die Oberflächenentwicklung, das so genannte „Frontend“ und zum Zweiten in die Untergrundentwicklung, besser bekannt als „Backend“. Das Backend versorgt eine Webseite hauptsächlich mit Daten, bsw. aus einer Datenbank. Das Frontend in der Webentwicklung ist zuständig für die technische Umsetzung der Konzeption einer Webseite. Hier wird die Struktur und das Aussehen entwickelt, das „User Interface“. Die grundlegenden Techniken zur Entwicklung von Webseitenoberflächen sind HTML und CSS, welche für die Struktur und das Styling verantwortlich sind.

3.2 Ansätze der Frontend-Entwicklung

Ein Frontend-Projekt allein durch HTML, CSS und JavaScript zu entwickeln kann sehr zeitaufwändig und umständlich werden. Da sich einige Elemente auf allen oder mehreren Unterseiten wiederholen, z.B. die Navigation oder der Footer, müsste eine Änderung an diesen Elementen auf jeder einzelnen Seite vorgenommen werden. Um diesen Aufwand zu minimieren hat sich im Laufe der Jahre eine modulare Herangehensweise etabliert. Hierbei werden sich wiederholende Elemente einer Webseite in einzelne Dateien ausgelagert und später zu einer Seite zusammengefügt. So bedarf es bei Änderungen nur noch eine einzige Datei zu bearbeiten.

Im Folgenden werden zwei Vorgehensweisen für eine modulare Bearbeitung anhand einer kleinen Beispielseite näher beschrieben.

Die Beispielseite bekommt zwei Unterseiten mit einem Menü und etwas Inhalt.

3.2.1 Mit PHP

In der ersten vorgestellten Methode werden die einzelnen Dateien via PHP zusammengefügt. Zu Projektbeginn wird eine Ordnerstruktur angelegt. In diesem Fall liegen im Wurzelverzeichnis folgende Dateien und Ordner:

```
- index.php
- ueber.php
- content.php
      - index.php
      - ueber.php
- elements.php
      - head.php
      - nav.php
- css
      - style.css
- img
      - berlin.png
```

Quellcode 3.1: Ordnerstruktur in der Frontend-Entwicklung mit PHP

Die beiden PHP-Dateien *index.php* und *ueber.php* auf der obersten Ebene werden direkt im Browser angesteuert. Beide besitzen den selben Aufbau:

```
1  <?php
2      $title = "Home";
3  ?>
4  <!DOCTYPE html>
5  <html>
6     <head>
7           <?php include "elements/head.php";?>
8     </head>
9     <body>
10          <?php include "elements/nav.php";?>
11
12          <div class="main">
13                <?php include "content/index.php";?>
14          </div>
15    </body>
16 </html>
```

Quellcode 3.2: *index.php*

Quellcodebeschreibung

Quellcode wird in farblich unterlegten Kästen gezeigt. Die darunter folgenden Absätze beschreiben bestimmte Zeilen dieses Codes.

7, 10 = Zeile 7 und 10

2 In der Zeile 2 wird eine PHP-Variable *$title* erzeugt und der Titel der Unterseite zugewiesen. Im Fall der zweiten Unterseite, *ueber.php*, wird der Variablen der Wert „Über“ zugewiesen. Darunter folgt das Markup für eine einfache HTML-Seite.

7, 10 Zwei Elemente tauchen auf jeder Unterseite gleichermaßen auf, der HTML-„head“ und die Seitennavigation. Diese beiden wurden in den Ordner *elements* ausgelagert. Um diese Parts auf einer Seite auszugeben, müssen diese durch ein „include“ an der richtigen Stelle eingebunden werden.

In der *head.php* werden der Titel der Seite und die Zeichencodierung festgelegt sowie die CSS-Datei eingebunden (Quellcode 3.3).

```
1 <title>MySite - <?php echo $title; ?></title>
2 <meta charset="utf-8">
3 <link rel="stylesheet" href="css/style.css">
```

Quellcode 3.3: *head.php*

In der Datei *nav.php* befindet sich das Menü der Seite.

```
 1 <nav>
 2    <ul>
 3          <li>
 4                <?php if($title == "Home"): ?><a class="active">
 5                <?php else: ?><a href="index.php">
 6                <?php endif; ?>
 7                Home
 8                </a>
 9          </li>
10          <li>
11                <?php if($title == "Über"): ?><a class="active">
12                <?php else: ?><a href="ueber.php">
13                <?php endif; ?>
14                Über
15                </a>
16          </li>
17    </ul>
18 </nav>
```

Quellcode 3.4: *nav.php*

4, 11 Hier wird durch eine *if-else*-Abfrage das *$title*-Attribut abgefragt. Dadruch wird ermittelt, ob gerade die Seite *index.php* oder *ueber.php* im Browser angezeigt wird und welches Menü-Item dementsprechend die CSS-Klasse „active“ bekommt.

Der Übersichtlichkeit halber gibt es den Ordner *content* in dem der jeweilige Inhaltsbereich der Unterseiten liegt.

In der Datei *content/index.php* gibt es eine Überschrift, die den Inhalt der *$title*-Variable anzeigt, ein Bild und Text.

Wird nun die *index.php* im Browser aufgerufen erscheint folgende Seite (Abbildung 3.1 und Quellcode 3.5):

Abbildung 3.1: Quelltext der im Browser angezeigten Seite

Quellcode 3.5: Quelltext der im Browser angezeigten Seite

```
<!DOCTYPE html>
<html>
<head>
   <title>MySite - Home</title>
   <meta charset="utf-8">
   <link rel="stylesheet" href="css/style.css">
</head>
<body>
   <nav>
         <ul>
               <li><a class="active">Home</a></li>
               <li><a href="ueber.php">Über</a></li>
         </ul>
   </nav>
   <div class="main">
         <h1>Home</h1>
         <img src="img/berlin.png" alt="" width="200">
         <p>
               Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
               sed diam nonumy eirmod tempor invidunt
               ut labore et dolore magna aliquyam erat,
               sed diam voluptua.
               At vero eos et accusam et justo duo dolores et ea rebum.
               Stet clita kasd gubergren,
               no sea takimata sanctus est Lorem ipsum dolor sit amet.
         </p>
   </div>
</body>
</html>
```

Frontend-Projekte mit PHP zu realisieren ist eine sehr simple Methode, da es dafür nur einen lokalen Server mit PHP-Unterstützung und wenig Zeit für Projekteinrichtung benötigt. Da allerdings erst noch die einzelnen Teile zu einer Webseite zusammengesetzen werden müssen, braucht der Browser etwas länger, um die Seite anzuzeigen.

3.2.2 Mit Static Site Generatoren

Eine weitere Möglichkeit für die Umsetzung von Frontend-Projekten ist die Entwicklung mit einem Static Site Generator (SSG). Vom Prinzip ist es das Gleiche wie die Herangehensweise mit PHP, nur wird in diesem Fall die Webseite nicht erst zusammengesetzt, wenn jemand die Webseite aufruft, sondern schon bevor sie überhaupt auf den Server geladen wird. Am Ende der Entwicklung setzt der Static Site Generator alle einzelnen Module zusammen und heraus kommen statische HTML-Seiten, die auf den Server geladen werden.

Zum direkten Vergleich wird im Folgenden die kleine Seite als ein Static Site Generator-Projekt umgesetzt.

```
- Gruntfile.js
- package.json
- src
      - assets
            - css
                  - style.css
            - img
                  - berlin.png
      - data
            - site.yml
      - templates
            - layouts
                  - default.hbs
            - pages
                  - index.hbs
                  - ueber.hbs
            - partials
                  - head.hbs
                  - nav.hbs
```

Quellcode 3.6: Ordnerstruktur in der Frontend-Entwicklung mit einem Static Site Generator

Für ein SSG-Projekt wird ein Task Runner benötigt. Dieser wird unter Punkt 3.4 genauer erklärt. Bis hier ist wichtig zu wissen, dass der Task Runner in der Datei *Gruntfile.js* definiert wird und dafür sorgt, dass der Static Site Generator seine Aufgabe erledigt. Alles was die eigentliche Webseite betrifft, befindet sich im Ordner *src*. In *assets* liegen CSS- und Bild-Dateien. Im Ordner *data* liegen Dateien, die Inhalte wie Texte, Überschriften etc., enthalten können. Aktuell steht in *site.yml* nur „title: MySite“. Diese Information wird später in der *head.hbs* noch aufgegriffen. Ein Static Site Generator arbeitet eng zusammen mit einer Template-Engine. In diesem Fall wird „Handlebars“ verwendet, daher die Endung „.hbs“ für die Dateien, die das Markup enthalten. Die Template-Engine sagt dem Static Site Generator welcher Content bzw. welche weitere Template-Dateien wo eingefügt werden sollen.

Das eigentliche HTML-Markup steckt im *templates*-Ordner. Die *default.hbs* im Ordner *layouts* bestimmt den generellen Seitenaufbau (Quellcode 3.7).

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      {{> head}}
5    </head>
6    <body>
7      {{> nav}}
8
9      <div class=”main”>
10       {{> body}}
11     </div>
12   </body>
13 </html>
```

Quellcode 3.7: *default.hbs,* gibt die Struktur der Webseite an

4 Das Pendant zum „include“ von PHP sind in diesem Fall die Platzhalterausdrücke der Template-Engine

Handlebars, sie stehen in doppelt geschweifte Klammern. Durch die Kombination des „>“-Zeichen mit dem Dateinamen einer weiteren *.hbs*-Datei wird angegeben, dass genau diese Datei hier eingebunden wird. Assemble interpretiert diese Ausdrücke und fügt die entsprechenden Inhalte ein.

4, 10 In den ersten beiden Fällen werden an diesen Stellen *head* (für *partials/head.hbs*) und *nav* (für *partials/nav.hbs*) eingesetzt.

In der *head.hbs* werden nochmals drei Teile eingefügt.

```
1 <meta charset="UTF-8">
2 <meta name="viewport" content="width=device-width, initial-
3 scale=1">
4 <title>{{title}} | {{site.title}}</title>
5
6 <!-- Custom styles for this template -->
7 <link href="{{assets}}/css/style.css" rel="stylesheet">
8
```

Quellcode 3.8: *head.hbs*

4 Ein Ausdruck ohne „>“-Zeichen steht für einen Platzhalter von einfachen Content. Der Inhalt der Variable *title* wird in jeder Datei im *pages*-Ordner festgelegt und an dieser Stelle über die geschweiften Klammern eingefügt. „{{ site.title }}“ zeigt auf das *title*-Attribut, dass in der *site.yml*-Datei definiert wurde.

7 „{{assets}}“ fügt den richtigen Pfad zur CSS-Datei ein.

Im Unterschied zu der PHP-Variante muss in der *nav.hbs* nur ein *li*-Element angelegt werden.

```
1 <nav>
2   <ul>
3       {{#each pages}}
4         <li>
5           <a{{#if this.isCurrentPage}} class="active"{{/if}}
6           href="{{relative dest this.dest}}">{{ data.title }}
7           </a>
8         </li>
9       {{/each}}
10
11   </ul>
12 </nav>
```

Quellcode 3.9: *nav.hbs,* enthält das Template für das Menü

3 Das #-Zeichen kennzeichnet einen ViewHelper, eine Funktion von Handlebars. Durch die *each*-Schleife wird für jede Datei im *pages*-Ordner ein Listenelement („
...
“) angelegt.

5 Zusätzlich wird durch eine *if*-Abfrage untersucht, ob diese Seite die Aktuelle ist, ist dies der Fall, bekommt der Link („<a>...“) des Listenelementes noch die CSS-Klasse „active“ zugefügt.

6 „{{ data.title }}“ fügt an dieser Stelle den Titel der Seite ein, der selbe, der auch in der *head.hbs* eingefügt wurde.

Bei „{{> body }}“ in der *default.hbs* verhält es sich etwas anders. Hier durchläuft der Static Site Generator jede Datei, die im Ordner *pages* liegt und erzeugt dafür jeweils eine eigene HTML-Seite.

Der Ordner *pages* ist mit dem *content*-Ordner des PHP-Projektes zu vergleichen. Hier liegen die Inhalte der beiden Unterseiten *Index* und *Über*.

```
---
title: Home
img: assets/img/berlin.png
---

   <h1>{{ title }}</h1>
   <img src="{{ img }}" alt="" width="200">
   <p>
         Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
         sed diam nonumy eirmod tempor invidunt ut labore et dolore
         magna aliquyam erat, sed diam voluptua. At vero eos et
         accusam et justo duo dolores et ea rebum. Stet clita kasd
         gubergren, no sea takimata sanctus est Lorem ipsum dolor
         sit amet.
   </p>
```

Quellcode 3.10: *index.hbs,* gibt Inhalte der Unterseite *Home* Webseite an

1–4 Hier werden einige Informationen festgelegt, die in den vorherigen Dateien bereits eingesetzt wurden und auch noch in dem darunter liegenden Code eingefügt werden.

Sind nun alle Layouts, Seiten und Module entwickelt, beginnt die Arbeit des Static Site Generators. Er wird durch den Task Runner aktiviert und setzt nun alle Teile zusammen und speichert diese als statische HTML-Seiten in einem automatisch angelegten Ordner *dist*:

```
- dist
      - assets
            - css
                  - style.css
            - img
                  - berlin.png
- index.html
- ueber.html
```

Quellcode 3.11: Ordnerstruktur, des von Assemble generierten Ordner *dist*

Wird nun die Seite „index.html“ im Browser aufgerufen, erscheint das selbe Ergebnis wie auch schon beim PHP-Projekt.

Diese Variante hat u.a. den Vorteil von geringeren Ladezeiten im Browser. Dafür ist das Aufsetzen eines solchen Projektes um einiges aufwendiger, da es neben dem Static Site Generator noch weitere Komponenten wie einen Package Manager, einen Task Runner und eine Template-Engine benötigt, welche im weiteren Verlauf noch vorgestellt werden. Da der Trend für statische Webseiten und Static Site Generators neu auflebt (Schwarz, 2016), soll dieser Ansatz für den in dieser Arbeit entstehenden Workflow angewandt werden.

3.3 Package Manager

Für einen Webentwickler beginnt die Arbeit immer mit der Einrichtung des Projektes. Das beinhaltet u.a. das Herunterladen von unterstützenden Web-Applikationen wie Bootstrap, jQuery, einem Task Runner, usw. Dieser doch etwas langwierige Prozess, jede Datei von der entsprechenden Webseite herunterzuladen, kann von einem Package Manager übernommen werden. Ein Package Manager ist ein Konsolen-Programm, dass sich um die Installation, Aktualisierung und Deinstallation von Web-Applikationen in Paketform kümmert (vgl. *wikipedia.org*).

Abbildung 3.2:
Logo des Package Managers npm (*npmjs.com*)

Im Rahmen dieser Arbeit wird der Node Package Manager (npm) verwendet. Für Entwickler ist npm eine Plattform zum Austausch von Code. Effizienter Code kann als ein Paket von Entwicklern auf diese Plattform für andere Entwickler verfügbar gemacht werden.

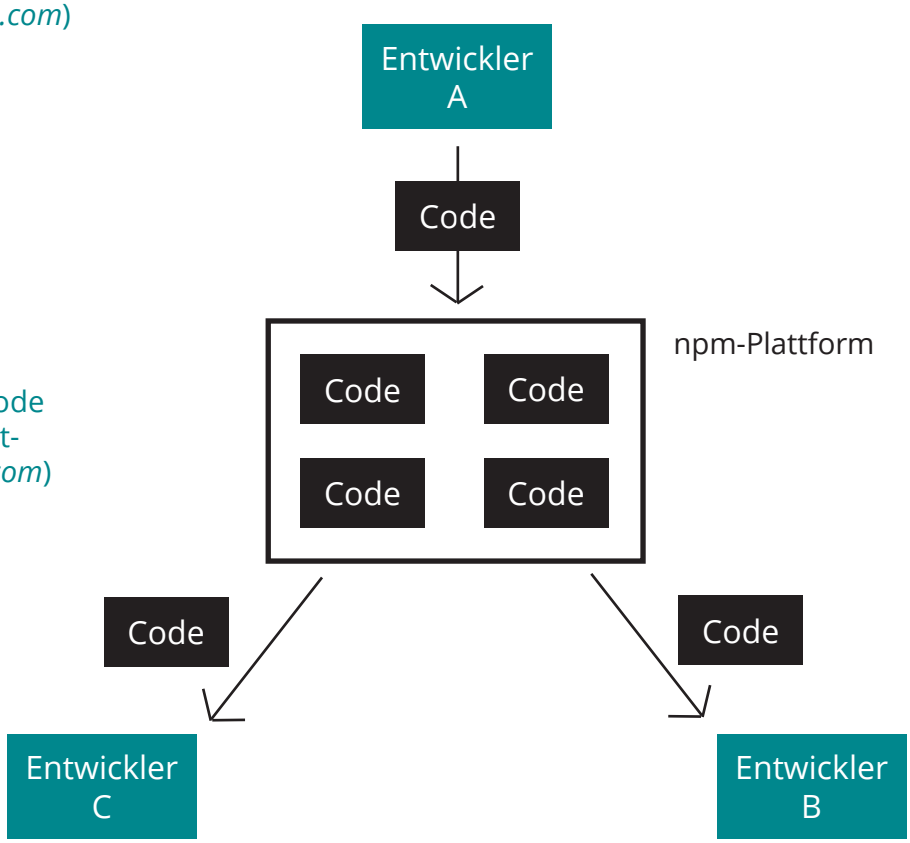


Abbildung 3.3:
Funktionsweise der Node Package Manager-Plattform (vgl. *docs.npmjs.com*)

Nach der Installation von npm auf dem eigenen Rechner kann das Herunterladen über die Konsole der Pakete erfolgen. Für eine lokale Installation von Paketen muss zunächst über die Konsole in den gewünschten Ordner navigiert werden. Zuerst wird ein neues Projekt angelegt, dafür wird der Befehl

```
npm init
```

ausgeführt. Dadurch wird die Datei *package.json* erstellt, welche Projektinformationen und die Abhängigkeiten des Projektes von anderen Paketen enthält.

Um Pakete zu installieren, wird der Befehl

```
npm install PAKETNAME --save-dev
```

ausgeführt. Nun beginnt der Node Package Manager das gewünschte Paket in den Ordner zu laden. Die dadurch entstandene Abhängigkeit von diesem Paket wird durch die Option „--save-dev“ in die *package.json* geschrieben.

3.4 Task Runner

Wie auch ein Package Manager soll ein Task Runner die Arbeit eines Entwicklers erleichtern, indem er immer wiederkehrende Aufgaben übernimmt. So zum Beispiel das Kompilieren von Less- in CSS-Dateien oder das Neuladen des Browsers bei geänderten Dateien.

Der in diesem Fall verwendete Task Runner ist Grunt.

Abbildung 3.4:
Logo des Task Runners Grunt (*bitballoon.com*)

Grunt wird mit Hilfe des Package Managers Node über folgenden Kommandozeilenbefehl in das neu angelegte Projekt integriert:

```
npm install grunt --save-dev
```

Nach der Installation wird eine Datei zur Konfiguration von Grunt und dessen Aufgaben angelegt, „Gruntfile.js“. Hier werden alle gewünschten Aufgaben definiert, die Grunt für den Entwickler übernehmen soll. Im Netz existieren viele Plugins, die Aufgaben übernehmen und in Grunt integriert werden können. Auch diese Plugins werden über npm installiert:

```
npm install grunt-contrib-PLUGINNAME
--save-dev
```

In der *Gruntfile.js* wird nun das Plugin und die Konfiguration dieser Aufgabe definiert und durch den Befehl

```
grunt TASKNAME
```

ausgeführt (Vgl. Ettisberger, 2015).

Auf der offiziellen Seite *gruntjs.com* stehen bereits 5.400 Plugins zur Verfügung (*http://gruntjs.com/plugins* [30.12.15]).

3.5 Static Site Generator

Wie schon unter Punkt 3.2.2 beschrieben, sorgt ein Static Site Generator für die Zusammensetzung von einzelnen Code-Fragmenten zu statischen HTML-Seiten.

Static Site Generatoren haben unter anderem auch den Zweck eine Frontend-Produktion bequemer zu gestalten, dessen Ergebnis dann an ein Content Management System gebunden wird. Für viele komplette Webprojekte würde eine statische Seite vollkommen ausreichen, da nicht ständig neuer Content generiert wird. Statische Webseiten haben ihre Vorteile gegenüber Webseiten mit Content Management Systemen. Sie arbeiten z.B. ohne Datenbank und serverseitiger Skriptsprache wie z.B. PHP, die jeweils eine große Angriffsfläche für Hacker bieten. Beim Aufrufen von dynamsichen Seiten muss die Seite erst durch Datenbankabfragen zusammengebaut werden, bevor sie angezeigt werden kann. Im Gegensatz dazu ist eine statische HTML-Seite bei Anfrage sofort bereit für die Anzeige. Dazu benötigen CMSe besondere Server-Anforderungen, was zu teureren Hostingpaketen führen kann. Ebenso werden für die Bearbeitung einer Website technische Kenntnisse verlangt. Müssen Änderungen an der Seite vorgenommen werden, geht das nur über den Entwickler oder unter der Verwendung eines kostspieligen CMS für SSE, z.B. Contentful. Ebenso sollten vor dem Projektstart einige Kriterien abgefragt werden, ob ein Static Site Generator überhaupt in Frage kommt:

- Soll es auf der Site eine Benutzerregistration geben?
- Soll es eine Kommentar- oder Suchfunktion geben?
- Müssen Formular-Daten von Nutzern auf dem Server gespeichert werden?

Werden diese Fragen mit „Ja“ beantwortet, fällt ein statisches Webprojekt raus. (Droste, 2016). Da in dieser Arbeit der Static Site Generator nur in der Frontend-Produktion zum Einsatz kommt, sind die Kriterien irrelevant.

Abbildung 3.5:
Logo des Static Site Generators Assemble (*assemble.io*)

Bei der Umsetzung des Frontend-Templates dieser Arbeit wird Assemble als Static Site Generator verwendet. Assemble wurde von Jon Schlinkert und Brian Woodward als Grunt-Plugin entwickelt und basiert auf JavaScript.

Das Starten des Prozesses zur Zusammensetzung aller Webseitenelemente kann ebenfalls durch den Task Runner übernommen werden. Durch

```
npm assemble --save-dev
```

wird Assemble in das Projekt eingebunden und durch einen Eintrag in die *Gruntfile.js*-Datei für Grunt verwendbar.

3.6 Template-Engines

Abbildung 3.6:
Logo der Template-Engine Handlebars (*handlebarsjs.com*)

Durch eine Template-Engine wird der Content von dem Layout und der Programmierlogik getrennt, ganz nach dem bekannten Model-View-Controller-Prinzip. Anstelle des Contents werden Ausdrücke der Template-Engine als Platzhalter in das Template (die View) eingesetzt. Für Assemble wird die Template-Engine Handlebars verwendet, dessen Platzhalter in doppelt geschweiften Klammern angegeben werden. Beispiele wurden bereits im Kapitel 3.2.2 aufgeführt.

3.7 Frameworks

Frameworks sind Bibliotheken, welche Funktionen zur erleichterten Programmierung von eigenen Anwendungen enthalten. In dieser Arbeit kommt jQuery zum Einsatz. jQuery ist die meistverwendete Opensource JavaScript-Bibliothek (*wikipedia.org*, 2016). Die vielen zusätzlichen Funktionen erleichtern die DOM-Manipulation und -Navigation, Event-Handling sowie Verwendung von Ajax und ermöglichen Animationen. Funktionen der jQuery-Bibliothek werden von jQuery-Objekten aufgerufen, die durch ein *$* gekennzeichnet sind (*api.jquery.com*, 2016, Lizenz: *jquery.org/license/*).

Abbildung 3.7:
Logo des Frameworks
jQuery
(*wikipedia.org*)

3.8 CSS-Präprozessoren

Schon bei mittleren Webprojekten können die CSS-Befehle ein enormes Ausmaß erreichen. Viele Selektoren wiederholen sich aufgrund von Schachtelungen der HTML-Tags. Um diesem Problem Abhilfe zu verschaffen, wurden CSS-Präprozessoren entwickelt. Sie fördern die Übersichtlichkeit, indem sie Verschachtelungen von CSS-Code zulassen. In diesem Projekt wird der Präprozessor Less verwendet.

```
1 /* -- logo css -- */
2 .logo {
3   display: inline-block;
4 }
5 .logo img {
6   height: 50px;
7   padding: 10px;
8   vertical-align: middle;
9 }
```

Quellcode 3.12: Beispielcode für CSS

```
20 /* -- logo in less -- */
21 .logo {
22   display: inline-block;
23   img {
24     height: 50px;
25       padding: 10px;
26       vertical-align: middle;
27   }
28 }
```

Quellcode 3.13: Der CSS-Code zu less-Code kompiliert

23 Der *logo*-Selektor braucht im Less-Code für den *img*-Tag, im Gegensatz zu Zeile 5 in Quellcode 3.12, nicht nochmal aufgeführt werden.

Neben der Schachtelung ermöglichen CSS-Präprozessoren u.a. noch den Einsatz von Variablen und Mixins. Diese enthalten Werte bzw. CSS-Code, die dann an vielen Stellen im Styling eingesetzt werden können. Zum Beispiel sind nützliche Variablen die Corporate Farben einer Webseite. Für sie wird jeweils eine Variable angelegt mit dem Hex-Code der Farbe. An jeder Stelle an der normalerweise der Hex-Code steht würde, wird nun der Name der Variabel eingetragen.

Ändert sich mitten im Projekt der Wert einer Variable, muss dieser für das ganze Projekt nur an einer Stelle angepasst werden.

Browser sind nicht in der Lage Präprozessor-Code zu interpretieren, weshalb ein Compiler zum Einsatz kommt, der den Less-Code in CSS-Code übersetzt und in ebensolch eine Datei schreibt.

Abbildung 3.8:
Logo des Präprozessors Less
(*commons.wikimedia.org*)

4 Content Management Systeme

Das vierte Kapitel befasst sich mit Content Management Systemen, ihrer Definition, aktuellen Statistiken, dem Vergleich von drei Systemen und kommenden Trends.

4.1 Definition

Webbasierte Systeme zur Verwaltung von Webseiten ohne Kenntniss von Webtechnologien wie HTML oder CSS nennen sich Content Management Systeme. Sie ermöglichen das Hinzufügen und Verändern von Inhaltselementen und Seiten und Hinzufügen und Editieren von Content auf einer Webseite und trennen dabei den redaktionellen Teil vom Layout. Ebenso verfügen viele CMSe über eine umfangreiche Medien- und Nutzerverwaltung. Personen mit Zugriffsberechtigung bekommen einen Anmeldeaccount mit bestimmten Rechten zugewiesen. Diese verschiedenen Nutzergruppen werden als „Rollen" bezeichnet. Die Rolle des Administrators ist zu allem berechtigt. Dazu kommen verschiedene Versionen der Rolle „Redaktuer", der je nach Abstufung zur Veröffentlichung, zum Verfassen, Hinzufügen, Editieren und Löschen von Elementen berechtigt sein kann (vgl. *netupdater.de*, 2016).

4.2 Statistiken

Laut neusten Statistiken von *w3techs.com* steckten am 01. Februar 2016 hinter 43.7% aller Webseiten im Internet ein Content Management System (Abbildung 4.2). Vor einem Jahr waren es noch 38,5% (Abbildung 4.1). Innerhalb eines Jahres ist damit die weltweite Nutzung von CM-Systemen um 5,2% gestiegen.

Deutliche Unterschiede ergeben sich aus dem Vergleich, welche Systeme in Deutschland und weltweit genutzt werden. In beiden Statistiken ist Wordpress unangefochtener Spitzenreiter mit einem aktuellen weltweiten Marktanteil von 59.4% aller Webseiten mit CMS.

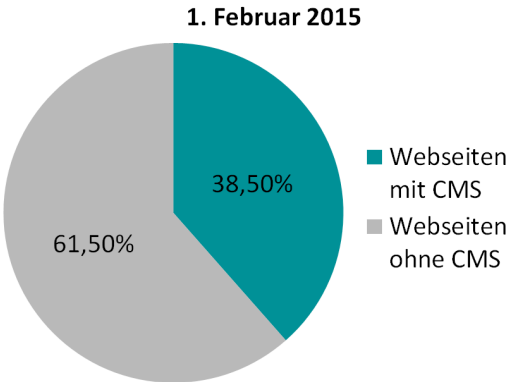


Abbildung 4.1:
Anteile von Webeiten mit und ohne CMS im Februar 2015 weltweit.

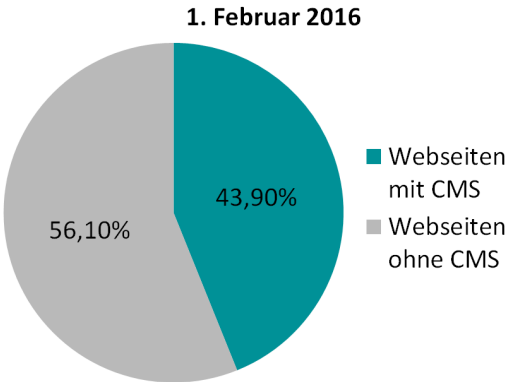


Abbildung 4.2:
Anteile von Webeiten mit und ohne CMS im Februar 2016 weltweit.

In Deutschland erreicht Wordpress (laut *webkalkulator.com*, 26.02.2016) einen Marktanteil von 51%. Dennoch ist aus dem Diagramm zu erkennen, dass der Markt in Deutschland noch breiter verteilt ist. Gerade Typo3 ist im weltweiten Vergleich in Deutschland beliebter (weltweit: 1,5%, deutschlandweit: 12%). Ebenso Contao (weltweit: 0,2%, deutschlandweit: 7%), was wohl daran liegt, dass dessen Herkunftsland Deutschland ist. (Abbildung 4.3)

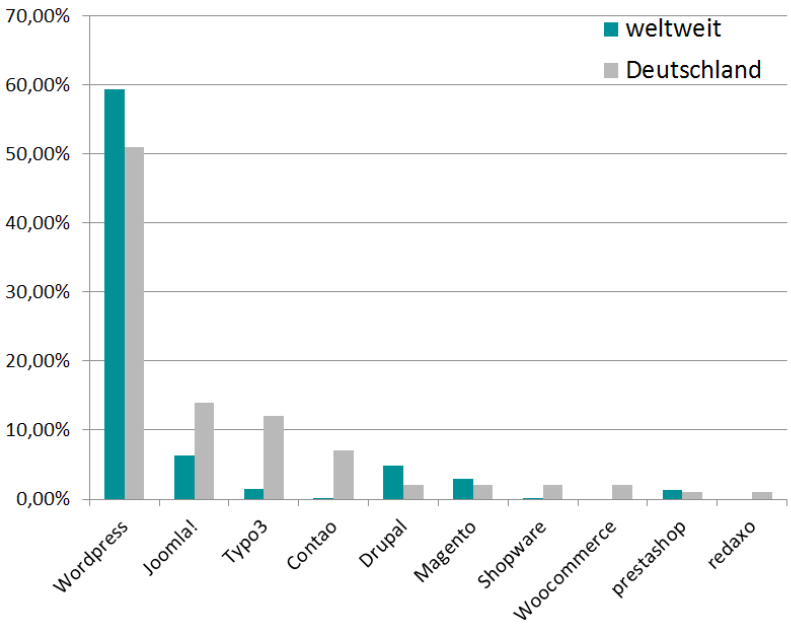


Abbildung 4.3:
Vergleich von Marktanteilen 2016 in Deutschland und weltweit zwischen den deutschlandweit zehn meistgenutzen CMS.

4.3 Vergleich von CMSen

Im Folgenden werden drei Content Management Systeme miteinander verglichen und schließlich eines ausgewählt, in welches das Frontend-Template aus Kapitel 6 integriert werden soll. In einem Test werden die Installation und das Backend jedes CMS untersucht. Zur Auswahl stehen das am weitesten verbreitete CMS Wordpress, dessen Nachfolger Joomla und das recht neue CMS TYPO3 Neos, welches eine vielversprechende Usability auszeichnet.

4.3.1 Wordpress

2003 begannen Matt Mullenweg und Mike Little die Entwicklung von der Webblog-Software Wordpress. 13 Jahre später wird Wordpress wöchentlich über 650.000 mal heruntergeladen und für Blogs, Webpräsenzen und Online-Shops verwendet. Es ist open source, basiert auf PHP und verwendet MySQL-Datenbanken. Es existiert eine mobile Version des CMS, wodurch sich Beiträge auch von unterwegs bearbeiten lassen. Für Wordpress existieren mehr als 20 Sprachpakete und es ist GNU lizensiert (*t3n.de*, 2016).

Test Wordpress

Wordpress lässt sich auf wordpress.org herunterladen. Von der offiziellen Webseite gibt es eine deutschsprachige Version (*de.wordpress.org*). Hier findet man u.a. den Download-Link, Themes, Plugins, einen Blog und Informationen zum Support. Durch den Klick des Download-Buttons auf der deutschen Startseite wird die aktuelle Version mit deutschem Sprachpaket heruntergeladen. Die *zip*-Datei wird in den Projektordner *htdocs* des lokalen Servers unter *wordpress* entpackt. Unter den Dateien befindet

sich eine *Liesmich.html*-Datei, die Tipps zu Installation und Aktualisierung, Systemanforderungen und eine Liste von Online-Ressourcen enthält.

Unter *local/phpmyadmin* wird eine neue Datenbank angelegt und die Informationen dazu in die Datei *wp-config.php* eingetragen. Anschließend wird im Browser die Adresse *local/wordpress/wp-admin/install.php* aufgerufen. Eine subjektive Begrüßungsnachricht erscheint mit einem Formular in das der Nutzer den Blogtitel, einen Benutzernamen und ein Passwort eingeben soll (Abbildung 4.4). Nach Eingabe und Bestätigung ist Wordpress nach nur wenigen Sekunden installiert und es kann sich im Backend angemeldet werden. Im Dashboard (Abbildung 4.5, S. 55), so wird das Backend von Wordpress genannt, fällt direkt eine Nachricht auf: es sei die Version Wordpress 4.4.2 verfügbar und das System solle jetzt aktualisiert werden, obwohl die heruntergeladene Version als 4.2.2 beschriftet wurde. Dazu befinden sich heruntergeladene Plugins und hochgeladene Themes aus einem früheren Wordpress-Projekt in der Installation, was darauf schließen lässt, dass nicht wie erwartet, die so eben heruntergeladenen Wordpressversion installiert wurde.

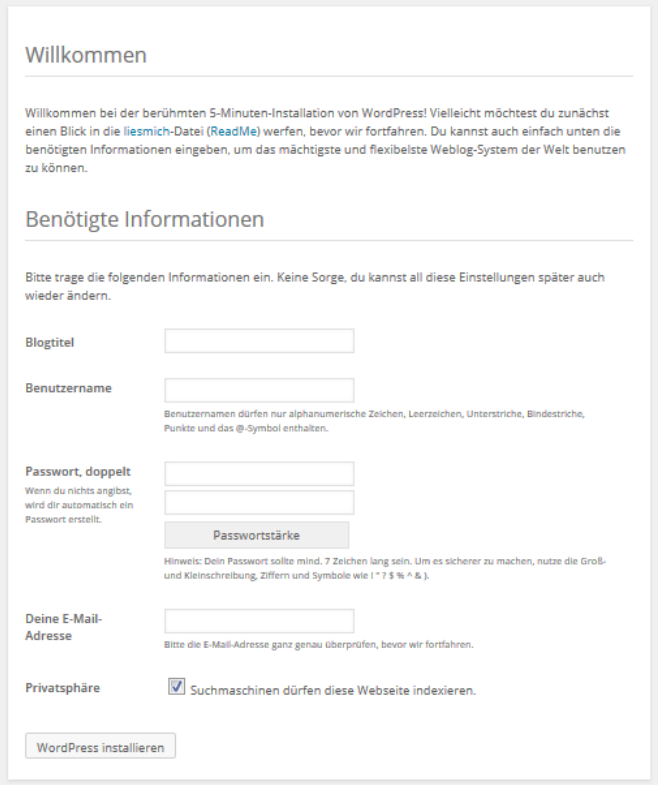


Abbildung 4.4: Installationsansicht Wordpress

Abgesehen davon ist das Dashboard recht geordnet. Links befindet sich ein Menü und im Zentrum eine Anzahl an Kästen, wovon Einer nächste, mögliche Schritte vorschlägt. Die Backend-Seiten für Seiten und Artikel zeigen eine Liste

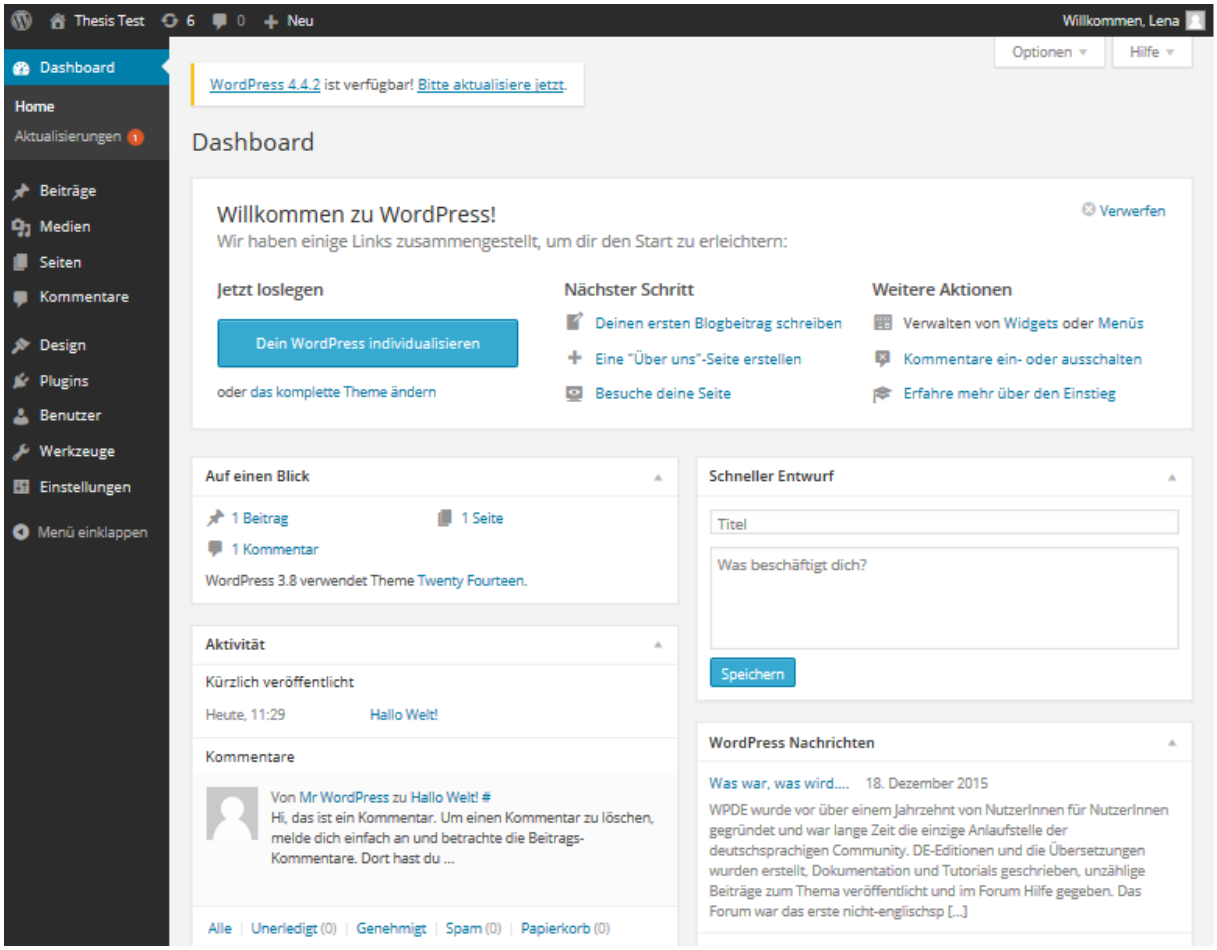


Abbildung 4.5: Backend Wordpress

der jeweiligen Elemente. Das Frontend, eine Beispiel-Seite, ist ein klassischer Blog mit einem Beitrag, Kategorien, Kommentaren, Archive und Meta-Tags. Unten links steht „Stolz präsentiert von Wordpress“. Um eine neue Unterseite anzulegen, wird im Dashboard im Menü der Punkt „Neu erstellen“ unter „Seiten“ ausgewählt. Nun steht die Eingabe eines Seitentitels und eines Textes in einen WYSIWYG-Editor zur Verfügung. Ebenso lassen sich der Zeitpunkt der Veröffentlichung und einige Attribute wie das Template oder das Elternelement wählen. Die Option auf ein Beitragsbild wird geboten. Bevor die neue Seite veröffentlich wird, kann eine Vorschau angezeigt werden. Neben dem Text und dem Titel wird noch eine Kommentarbox angezeigt. Im Frontend kann sich als

Benutzer angemeldet werden, dadurch erscheint neben den Beiträgen und in den Seiten ein „Edit“-Button, der zur „Bearbeiten“-Seite des jeweiligen Elements ins Backend weiterleitet.

Unter dem Menüpunkt *Design* im Dashboard lassen sich einige vorinstallierte Themes auswählen und anpassen. Direkt im Dashboard lassen sich neue Themes und Plugins suchen und installieren.

Zur abschließenden Betrachtung noch einige Vor- und Nachteile von Wordpress (Thattil, 2014):

Vorteile Wordpress

- Schnelle, einfache Installation
- Kurze Ladezeiten des Backends
- Viele frei erhältliche Themes und Plugins durch eine große, aktive Community
- Wordpress ermöglicht durch zusätzliche Plugins ein erfolgreiches SEO
- Optimale Blogsoftware

Nachteile Wordpress

- Keine Möglichkeit benutzerdefinierte Rollen anzulegen
- Mehrsprachigkeit wird erst durch die Installation eines Plugins möglich
- Durch die weite Verbreitung ein attraktives Ziel für Angreifer
- Es wird schnell deutlich, ob eine Seite über Wordpress erstellt wurde (über das Aussehen der Seite oder die Erwähnung von „Wordpress“ im Footer)

4.3.2 Joomla

Joomla ist ein Open Source Content Management System für kleine bis mittlere Unternehmen. Es basiert auf PHP und verwendet MySQL als Datenbank. Joomla ist GNU lizensiert. Im August 2005 trennten sich ein Großteil der Entwickler von dem Mambo-Projekt der australischen Firma Miro, um danach den Code unter dem Namen „Joomla“ selbständig weiter zu entwickeln. Drei Jahre später erschien dann die erste stabile Version des Systems, Joomla 1.5. Durch den offenen Quellcode kann Joomla von jedermann weiterentwickelt werden. (Vgl. *wikipedia.org*, 2016)

Test Joomla

Auch von Joomla lässt sich die aktuelle Seite auf einer offiziellen deutschsprachigen Webseite herunterladen (*joomla.de*). Im Gegensatz zu Wordpress muss ein deutsches Sprachpaket nachträglich in die Installation integriert werden. Die *.zip*-Datei wird wieder in ein neues Verzeichnis des *htdocs*-Ordner des lokalen Servers extrahiert. Im Internet finden sich Video- und Text-Anleitungen zur Installation von Joomla. Unter der URL *local/joomla* öffnet sich das Konfigurationsfenster (Abbildung 4.6). Bis zum Abschluss sind drei Schritte nötig. Wieder wird nach Benutzer und Datenbank gefragt, bevor die Installation gestartet wird. Auf der Abschlussseite wird die Option geboten, direkt ein Sprachpaket zu installieren. Nun kann der Anwender aus einer Liste zahlreicher Sprachen mehrere gleichzeitig auswählen und dadurch automatisch nachladen lassen. In einem weiteren Auswahlbildschirm lassen sich die Mehrsprachenfunktion, die Standard-Sprache für das Backend und für die Webseite auswählen.

Das Backend (Abbildung 4.7) besteht aus zwei Menüs, eines seitlich, eines am oberen Bildschirmrand und einem Content-Bereich, der eine Übersicht über neue und beliebte Artikel anzeigt. Allerdings reicht das Sprachenpaket nicht bis zu den Überschriften dieser Übersichtsboxen, sie sind in Englisch verfasst.

Abbildung 4.6: Installation Joomla

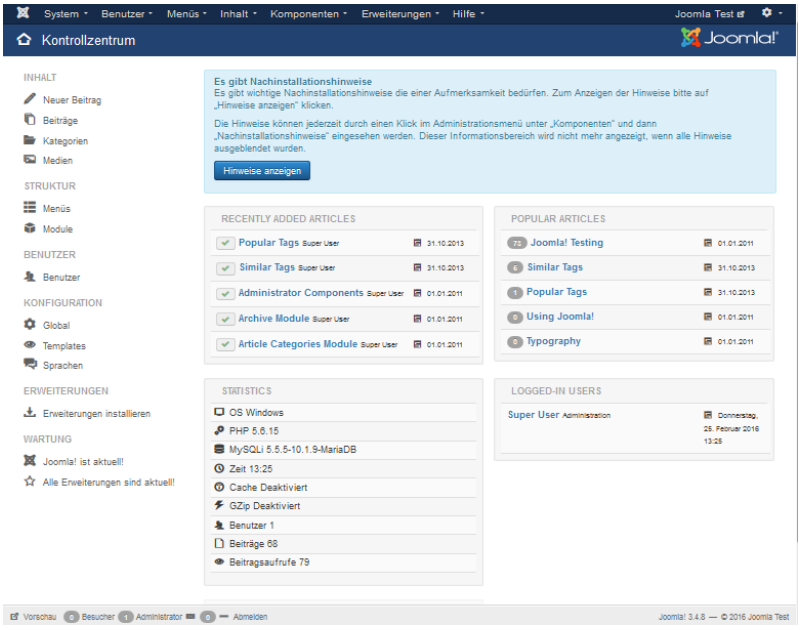


Abbildung 4.7: Backend Joomla

Es lassen sich das Template der Seite und des Backends anpassen. Je zwei vorinstallierte Themes stehen zur Auswahl. Weitere lassen sich wie bei Wordpress direkt im Backend aus einem Webkatalog laden. Alle Inhaltselemente der Webseite wie Beiträge, Komponenten, Menüs, Module, etc. sind in den jeweilige Unterseiten aufgelistet.

Auf den ersten Blick ist es schwer auszumachen, wie eine neue Seite angelegt werden kann. Für eine neue Unterseite muss ein neuer Eintrag in ein Menü angelegt werden. In einem Eingabefenster werden der Titel des neuen Eintrages und ein Menüeintrags-Typ angegeben. Zur Auswahl stehen zahlreiche Möglichkeiten wie „Alle Kategorien auflisten", „Haupteinträge", „Anmeldeformular", „Benutzerprofil" uvm. Der Typ gibt an, was auf der Seite angezeigt werden soll. Zusätzlich zu dem was durch den Typ angezeigt wird, ist es möglich der Seite Module zuzuweisen. Module sind, neben normalen Textbeiträgen, Inhaltselemente mit einem bestimmten Layout und einer vordefinierten Art von Content.

Joomla erlaubt eine umfangreiche Benutzerverwaltung, in der auch neue Rollen und Zugriffsebenen angelegt werden können. Unter der Systemkonfiguration findet der Nutzer einen Extraabschnitt für Suchmaschinenoptimierung und Einstellungsmöglichkeiten für jegliche Inhaltselemente.

Zur abschließenden Betrachtung noch einige Vor- und Nachteile von Joomla (Krahmer, 2016):

Vorteile Joomla

- Eingeschränkte benutzerdefnierte Rollen und Zugriffsebenen

- Durch Open Code kann Joomla von jedem weiterentwickelt werden
- Frontend-Editing von Beiträgen und Modulen möglich
- Funktionen für einen Online-Shop gegeben

Nachteile

- Kein intuitives Backend erschwert Anfängern den Einstieg
- Kleinere Community als Wordpress

4.3.3 TYPO3 Neos

Das jüngste Mitglied im Kreis der hier ausgewählten Content Management Systeme ist mit guten zwei Jahren seit der ersten stabilen Version TYPO3 Neos. Die Entwicklungen begannen bereits im Jahr 2006, ursprünglich um TYPO3 von grundauf zu modernisieren. Die Erneuerungen bildeten ein so großes Ausmaß, dass die Entwicklungen parallel zum herkömmlichen TYPO3 unter eigenem Namen weitergeführt wurden. Das große Merkmal des CMS ist das Editieren von Seiten und Inhalten. Im Gegensatz zu anderen System-Backends wird das Frontend im Backend angezeigt oder anders gesagt, das Backend wird auf das Frontend gelegt. Änderungen können direkt in der Seite vorgenommen werden. TYPO3 Neos basiert auf dem PHP Framework Flow und der Template-Engine Fluid (*mittwald.de/neos*, 2016).

Test TYPO3 Neos

Die Installation von TYPO3 Neos gestaltet sich durchaus aufwendiger als die seiner Vorgänger. Auf der offiziellen Seite (*neos.io*) gibt es eine englischsprachige Anleitung zum Herunterladen von Neos über die Konsole. Das

Herunterladen über ein Archiv wird nur für die ersten Tests mit Neos empfohlen. Für die Installation über die Konsole sind PHP und Composer nötig.

Für die Konfiguration von Nutzer und Datenbank ist es für Windows-Nutzer nötig den lokalen Server als Administrator auszuführen, da ansonsten nicht alle benötigten Dateien installiert werden können. Zurvor sollte ein neuer *virtual host* in der *apache.conf* eingetragen werden, in diesem Fall *neos.test*. Für das Setup wird die Seite *neos.test/setup/login* aufgerufen und sich mit einem automatisch generierten Passwort eingeloggt. Nach fünf Schritten in denen die Systemvoraussetzungen geprüft, die Datenbank angebunden, ein neuer Benutzer angelegt und ein Demo-Seitenpaket importiert werden, die aus unbekanntent Problemen mehrmals unter Veränderung einiger Variablen wiederholt werden müssen, kann sich für das Backend angemeldet werden.

Das Backend baut sich um das Frontend auf. Links werden zwei Strukturen angezeigt. Die obere zeigt die Seitenstruktur und die untere die jeweilige Contentstruktur. Die Inhaltselemente werden wie eine Ordnerstruktur geschachtelt. Je nach Einstellung können die Inhalte wie in Abbildung 4.8 direkt angeklickt und bearbeitet werden. Strukturelemente wie Reihen oder Spalten können durch ein Plus-Symbol hinzugefügt werden. Dazu wird ein „Create New"-Dialog angezeigt, der Elemente auflistet, die eingefügt werden können. Auf der rechten Seite wird der sogenannte „Inspektor" angezeigt. Er enthält Eigenschaften, die zur aktuell ausgewählten Komponente eingestellt werden können. Auffällig sind die langen Ladezeiten des Backends bei Änderungen.

Oben links kann ein Menü geöffnet werden, wodurch zu den Medien, zum Administrationsbereich usw. weitergeleitet wird.

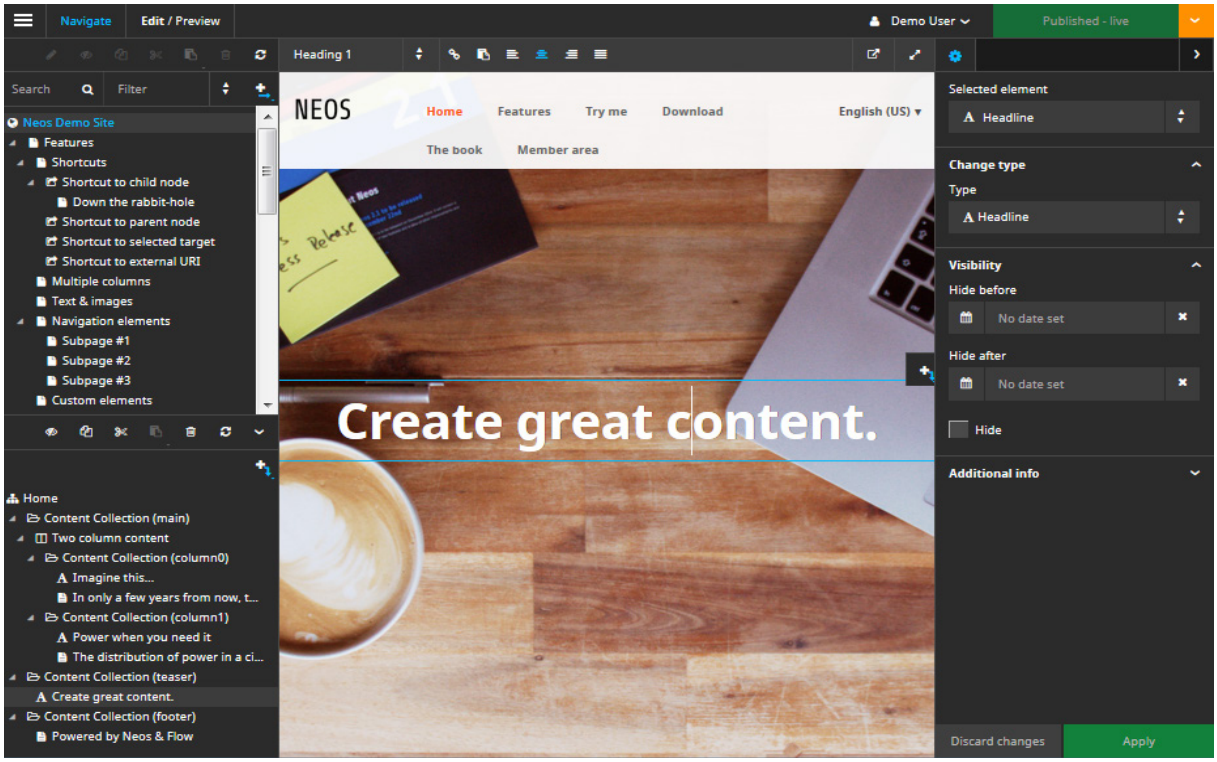


Abbildung 4.8: Backend TYPO3 Neos

Vorteile TYPO3 Neos

- hohe Benutzerfreundlichkeit
- intuitives Backend
- Frontend-Editing

Nachteile TYPO3 Neos

- Bisher nur eine kleine Community, die bei Fragen zur Verfügung steht
- Sehr lange Ladezeiten im Backend
- Bisher nur eingeschränkter Funktionsumfang
- Nur zwei Benutzerrollen zur Auswahl

Alle drei Content Management Systeme haben ihre Stärken und Schwächen. TYPO3 Neos ist jung, unentdeckt, aber dennoch vielversprechend. Einer der wichtigsten Aspekte ist die Usability für den Endbenutzer, hat er kein Gefühl für das Backend, läuft die Seite Gefahr „einzuschlafen“. Aus diesem Grund wird das Frontend-Template aus Kapitel 6 in TYPO3 Neos integriert.

4.4 Trends

Der Spitzenreiter Wordpress wird sich sobald nicht von seinem Thron stoßen lassen, zumal die Nutzerzahlen immer weiter steigen. TYPO3 ist ein beliebtes Content Management System in Deutschland, allerdings nur für große, umfangreiche Seiten geeignet, daher wird TYPO3 Neos als kleiner Bruder gute Chancen haben, sich in einigen Jahren am Markt für kleine und mittlere Seiten einen Namen zu machen. Vorraussetzung natürlich ist, dass die Entwickler motiviert weiterarbeiten und die Community wächst. Durch die hohe Benutzerfreundlichkeit könnte Neos eventuell Joomla einige Marktanteile streitig machen.

5 Konzeption eines allgemeinen Templates

Der Workflow zur Erstellung von Frontend-Templates mit anschließender Anbindung an ein CMS beginnt mit der Konzeption. Da zu diesem Zeitpunkt noch kein Kunde und dessen Wünsche berücksichtigt werden müssen, ist die Konzeption bei Zielgruppe und Stilrichtung völlig frei. Das Ziel der Konzeption ist ein schlichtes, aber dennoch ansprechendes, fluides Design mit einem vielfältigen Modulkatalog, durch den das Template auf verschiedene Kunden zutreffen kann, zu gestalten.

Allgemein wird ein Template inhaltstechnisch in das Layout und die Module aufgeteilt. Das Layout bestimmt die Positionen der Module und den Aufbau der Seite. Die Module bestimmen die Darstellung des Contents.

5.1 Moqups

Nach Zeichnungen auf Papier wird eine erste digitale Version erstellt. Im Rahmen dieser Arbeit wird das Online-Tool „Moqups“ verwendet.

5.1.1 Layout

Ganz allgemein formuliert, kann man das Web in drei grundlegende Inhaltstypen einteilen. Bei allen finden sich ein Header, ein Content-Bereich und ein Footer. Optional steht eine Sidebar, ab und an auch zwei, zur Verfügung, die je nach Belieben auf die linke (Abbildung 5.1) oder rechte Seite (Abbildung 5.2) positioniert wird. Platziert sich die Sidebar auf der linken Seite, enthält sie meist das Hauptmenü. Auf der rechten Seite jedoch enthält sie eher Inhaltsmodule wie eine Kontaktbox, Preview-Elemente oder z.B. Beiträge aus Social-Media-Kanälen.

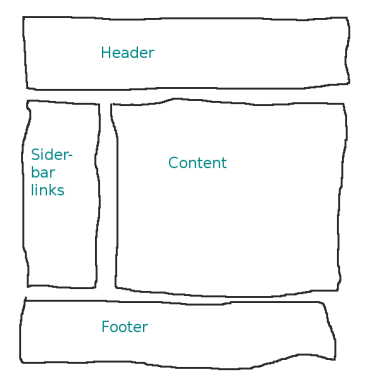


Abbildung 5.1:
Layout mit Sidebar links

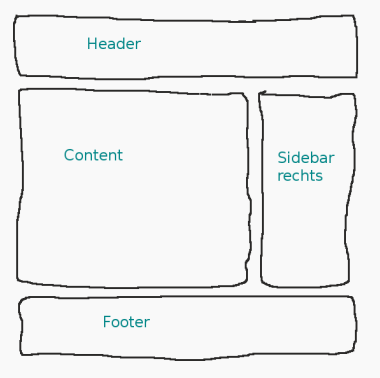


Abbildung 5.2:
Layout mit Sidebar rechts

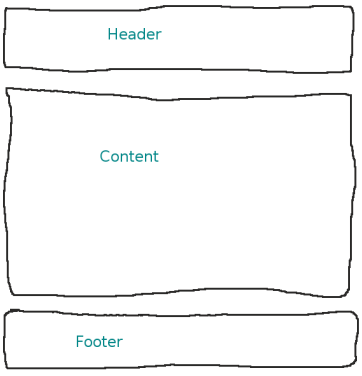


Abbildung 5.3:
Layout ohne Sidebar

Wird sich durch die zahlreichen Themes im Internet geklickt, wird allerdings mehr und mehr der Trend, das Layout ohne Sidebar anzulegen (Abbildung 5.3), deutlich. Hier befindet sich im Header das Hauptmenü und meist

ein großes Bild, einzeln oder als Slideshow. Im Content-Bereich werden die Module verteilt und im Footer findet sich häufig eine zweite Navigation, weitere Informationen oder auch Social-Media-Beiträge oder -Links. Der Vorteil der Stapelung ist die Übersichtlichkeit. Scrollt der User durch die Seite befindet sich in diesem Fall nur ein Content-Bereich in seinem Blickfeld. Er wird nicht durch zusätzliche Inhalte in der rechten Spalte abgelenkt.

Das Layout dieses Projektes bekommt einen schlichten Aufbau „Header - Content - Footer“ (Abbildung 5.4). Im Header werden das Logo und das Hauptmenü platziert. Der Content-Bereich wird mit Blöcken befüllt, die jeweils ein oder zwei Module enthalten können. Am Ende der Seite sitzt der Footer, dieser wird häufig mit Kontaktinformationen oder Links zu Unterseiten befüllt. In dieser Projektphase steht diesbezüglich noch keine Entscheidung.

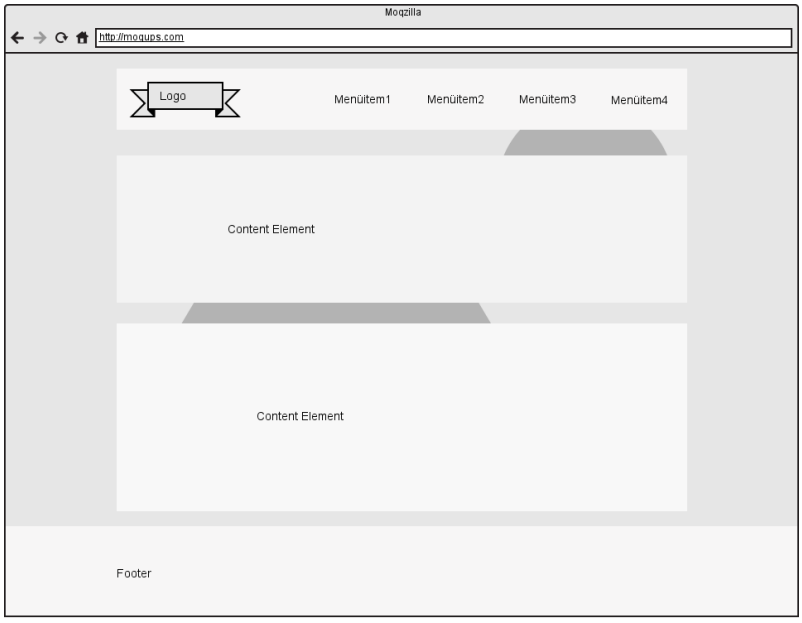


Abbildung 5.4: Layout des Templates als WIreframe

5.1.2 Module

Für den Inhalt werden einige häufig im Internet vorkommende Module ausgewählt:

- einen einfachen Artikel, mit Überschriften, Bild und Text
- ein Preview-Modul, mit Bild, einer Überschrift, weitere Infos und ein Button der zu mehr Content führt
- ein Icon-Modul, mit Icon und kurzem Text
- eine Bildergalerie mit Vorschau-Element
- eine interaktive Karte für Standorte, mit Anzeige der Kontaktdaten
- ein Kalender-Modul
- ein Kontaktformular
- ein großer, alleinstehender Schriftzug
- ein Accordion
- und ein Tab-Modul.

5.2 Konkretes Design mit Photoshop

Nach der Erstellung des Grobkonzeptes wird nun die realistische Gestaltung des Templates auf Grundlage der Moqups umgesetzt. Das Template wird mittels Photoshop in einer Desktopversion und einer mobilen Version visualisiert.

5.2.1 Layout

Die Farb- und Effektgestaltung wird an Googles Material Design angelehnt. Nur angelehnt, da sonst der Wiedererkennungswert der Seite sinkt und ein Nutzer an das typi-

sche Design von vielen anderen Seiten erinnert wird.

Schrift

Als Schriftart wurde Open Sans gewählt. Es ist eine weit ausgebaute, leichte, serifenlose Schrift, die der im Material

Light 300
Grumpy wizards make toxic brew for the evil Queen and Jack.

Normal 400
Grumpy wizards make toxic brew for the evil Queen and Jack.

Semi-Bold 600
Grumpy wizards make toxic brew for the evil Queen and Jack.

Bold 700
Grumpy wizards make toxic brew for the evil Queen and Jack.

Extra-Bold 800
Grumpy wizards make toxic brew for the evil Queen and Jack.

Abbildung 5.5: Layout des Templates als WIreframe

Design Guide neben Roboto empfohlenen Schriftart Noto (*google.com/design, 2015*) sehr ähnelt (*typografie.info*).

Farbe

Der Styleguide empfiehlt eine Hauptfarbe und eine Kontrastfarbe und unterstützt bei der Wahl mit einem Farbrad (Abbildung 5.6).

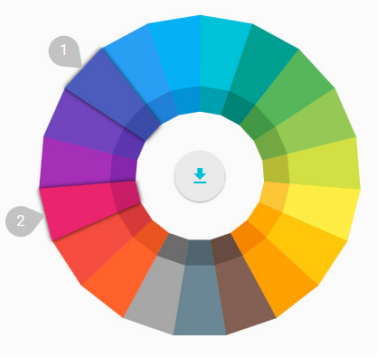


Abbildung 5.6: Farbrad zur Auswahl zusammen passender Farben, Material Design Styleguide (*getmdl.io*)

	R: 0 G: 183 B: 156	Kontrastfarbe für aktiven Menüpunkt, Links, Buttons, Hervorhebungen, ...
	R: 68 G: 122 B: 126	Hauptfarbe für das Menü und den Footer
	R: 79 G: 79 B: 76	Schriftfarbe

Die Farben sind nur beispielhaft für die Demoseite, um das Template einem Kunden zu präsentieren und können je nach Wunsch des Kunden geändert werden.

Icons

Die Icons kommen aus der Icon-Bibliothek des Material Design Styleguides von Google und sind frei nutzbar. (*design.google.com/icons*)

Bilder

Alle Bilder stammen aus der Bilddatenbank *pixabay.com* und sind frei von Urheberrechten.

Seitenaufbau

Der Hintergrund bekommt ein bildschirmfüllendes Foto, wobei bei dessen Wahl einige Dinge beachtet werden sollten. Das Bild dient nur als Hintergrundbild und soll den Fokus nicht auf sich lenken. Der Großteil des Bildes wird vom Content der Seite verdeckt sein. Die zwei Layout-Komponenten Header und Content bekommen eine beschränkte Breite, der Footer erstreckt sich über den kompletten Bildschirm. Alle Komponenten werden von einem leeren Abstand voneinander getrennt. Für die Module steht die Wahl, ob sie einen weißen oder gar keinen Hintergrund bekommen. Damit dabei dennoch der Kontrast zwischen Hintergrundbild und Content nicht leidet, wird generell eine weiße, leicht transparente Fläche eingefügt.

5.2.2 Module

Die Module werden in Reihen platziert. Ein Modul nimmt entweder die ganze Reihe in Anspruch oder teilt sie sich gleichmäßig mit einem weiteren Modul.

Die Header-Leiste (Abbildung 5.7) enthält links das Logo und rechts die Seitennavigation. Ein aktiver Menüpunkt bekommt einen 5px breiten unteren Rahmen in der vorab definierten Kontrastfarbe. Scrollt der Nutzer in der Seite nach unten, bleibt der Header am oberen Bildrand „kleben“. In der mobilen Version (Abbildung 5.8) wird das Menü eingeklappt und als „Hamburger“-Icon dargestellt. Öffnet man das mobile Menü durch einen Klick auf das Icon, dreht es sich um 90 Grad und die Menüpunkte werden aufgefahren.

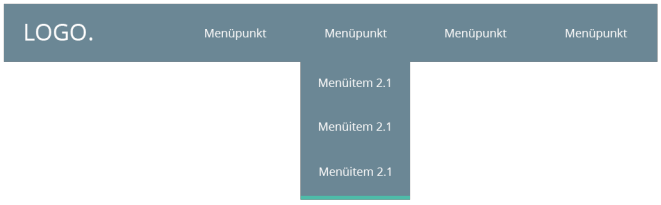


Abbildung 5.7: Desktop-Menü mit offnem Untermenü

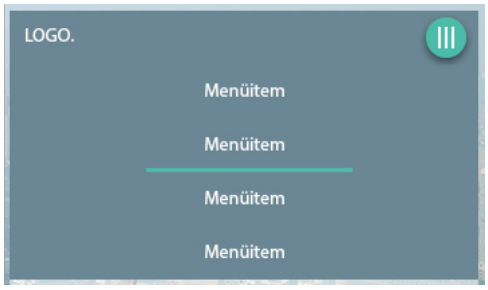


Abbildung 5.8: Mobiles Menü offen

Das Modul in Abbildung 5.9 kann eingesetzt werden, um einen Leitspruch, ein Zitat oder eine andere Information groß auf der Seite zu platzieren.

Abbildung 5.9: Modul: Große Überschrift

In dem Modul „Icon-Cards“ (Abbildung 5.10) stehen einfache, kurze Texte mit einem größeren Icon darüber. Solche Module werden häufig für die Leistungen oder die Werte des Unternehmens verwendet. Wird der Bildschirm kleiner, rutschen die Texte mit Icon nach und nach untereinander.

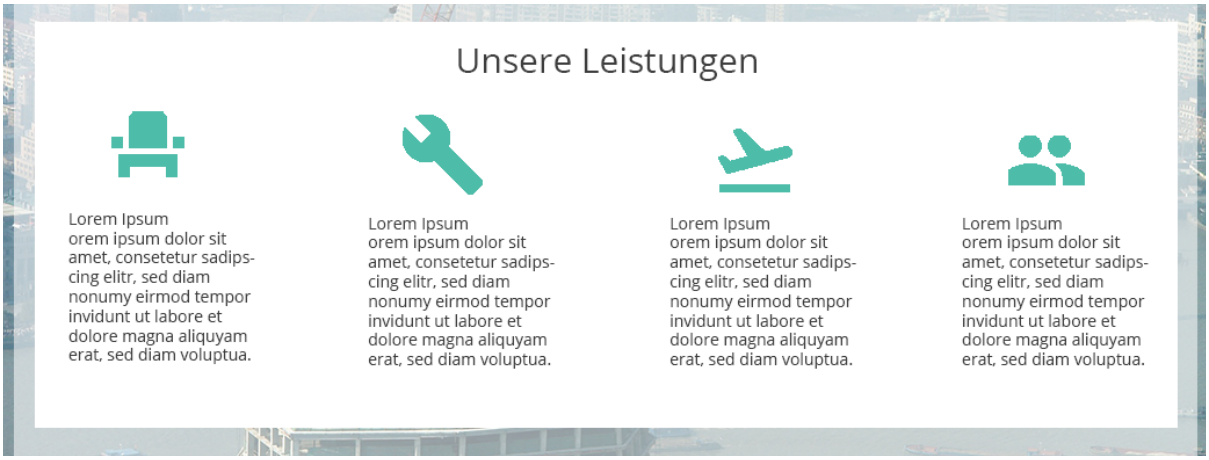
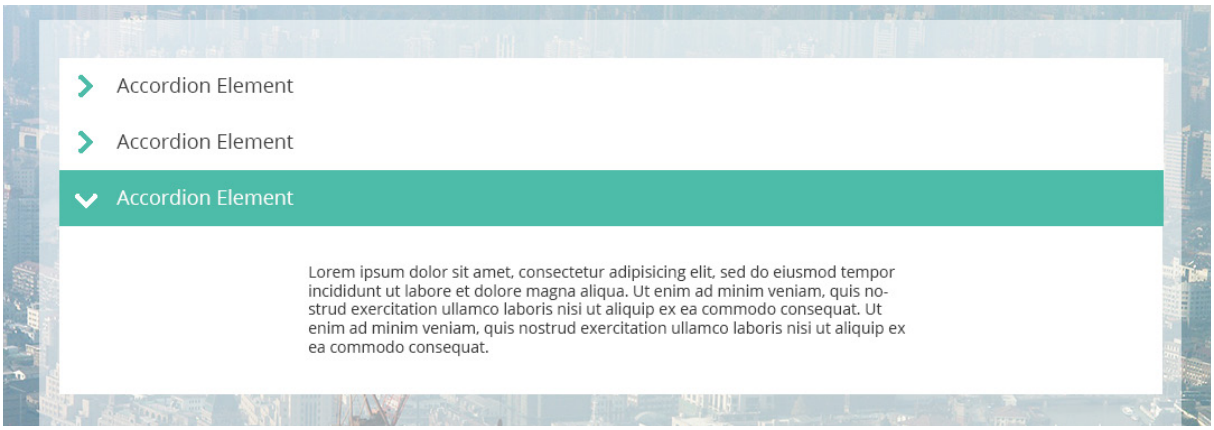


Abbildung 5.10: Modul: Icon-Cards

Das Accordion (Abbildung 5.11) besteht aus Elementen mit einer Überschrift und Content. Standartmäßig ist der Content nicht sichtbar, erst bei einem Klick auf die Überschrift wird der zugehörige Inhalt aufgefahren und ein eventuell anderer offener Content wird wieder eingefahren. Hier könnten beispielsweise auch die Leistungen, die Produkte oder die Angestellten beschrieben werden.

Abbildung 5.11:
Modul: Accordion



Das Preview-Modul (Abbildung 5.12) enthält einzelne Blöcke, die als Vorschau für News, Artikel, Events, etc. dienen können. Der Button mit dem „+“-Symbol führt zu weiteren Informationen. Diese können auf einer neuen Seite stehen oder aber auch direkt in dem weißen Teil des Blockes. Ist dies der Fall, wird durch den Klick auf den „+“-Button das weiße Kästchen bis an den oberen Rand des Blockes gefahren. Die Anzahl der Blöcke ist unbeschränkt. Die Anzahl der sichtbaren Blöcke ist von der Bildschirmbreite abhängig. Um die übrigen Blöcke zu sehen, klickt man auf die Buttons rechts und links unten im Bild. Das verschiebt die Reihe um eine Blockbreite.

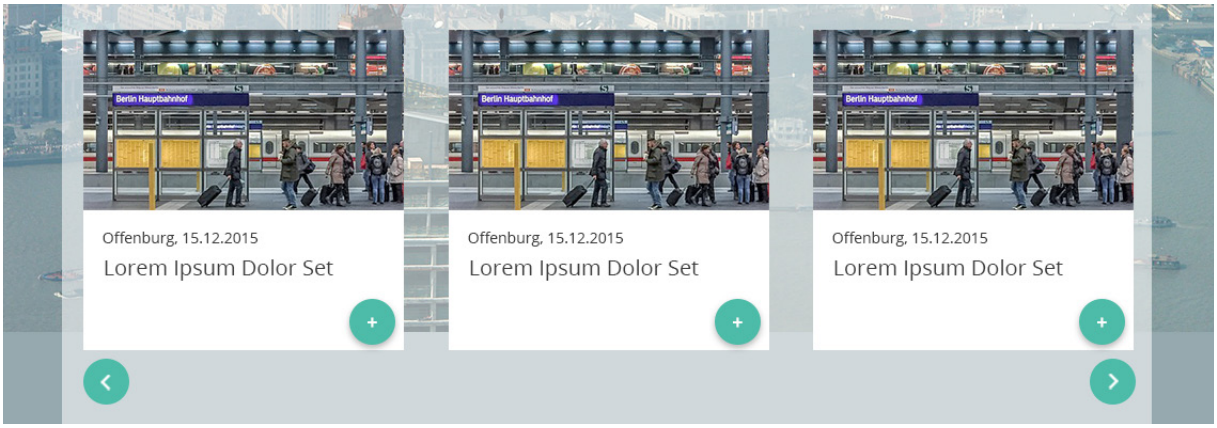


Abbildung 5.12: Modul: Preview

Die Galerie-Vorschau (Abbildung 5.13) zeigt eine beschränkte Auswahl an Bildern aus einer Bildergalerie an. Der „+“-Button unten rechts leitet zu der Galerie weiter.

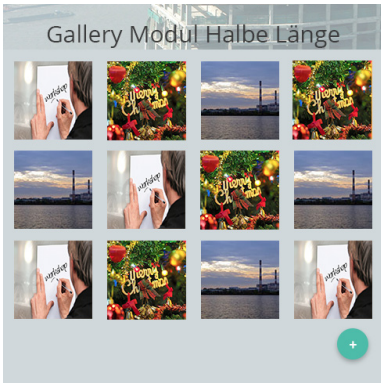


Abbildung 5.13:
Modul: Galerie-Vorschau

Durch das simple Formular, in Abbildung 5.14 (unten), kann durch Angabe seines Names, seiner E-Mail-Adresse oder Telefonnummer und einer Nachricht Kontakt zum Webseitenbetreiber aufgenommen werden.

Im Kalender-Modul, in Abbildung 5.14 (oben), bekommt der Kunde die Möglichkeit beispielsweise Veranstaltungen oder Betriebsferien anzugeben.

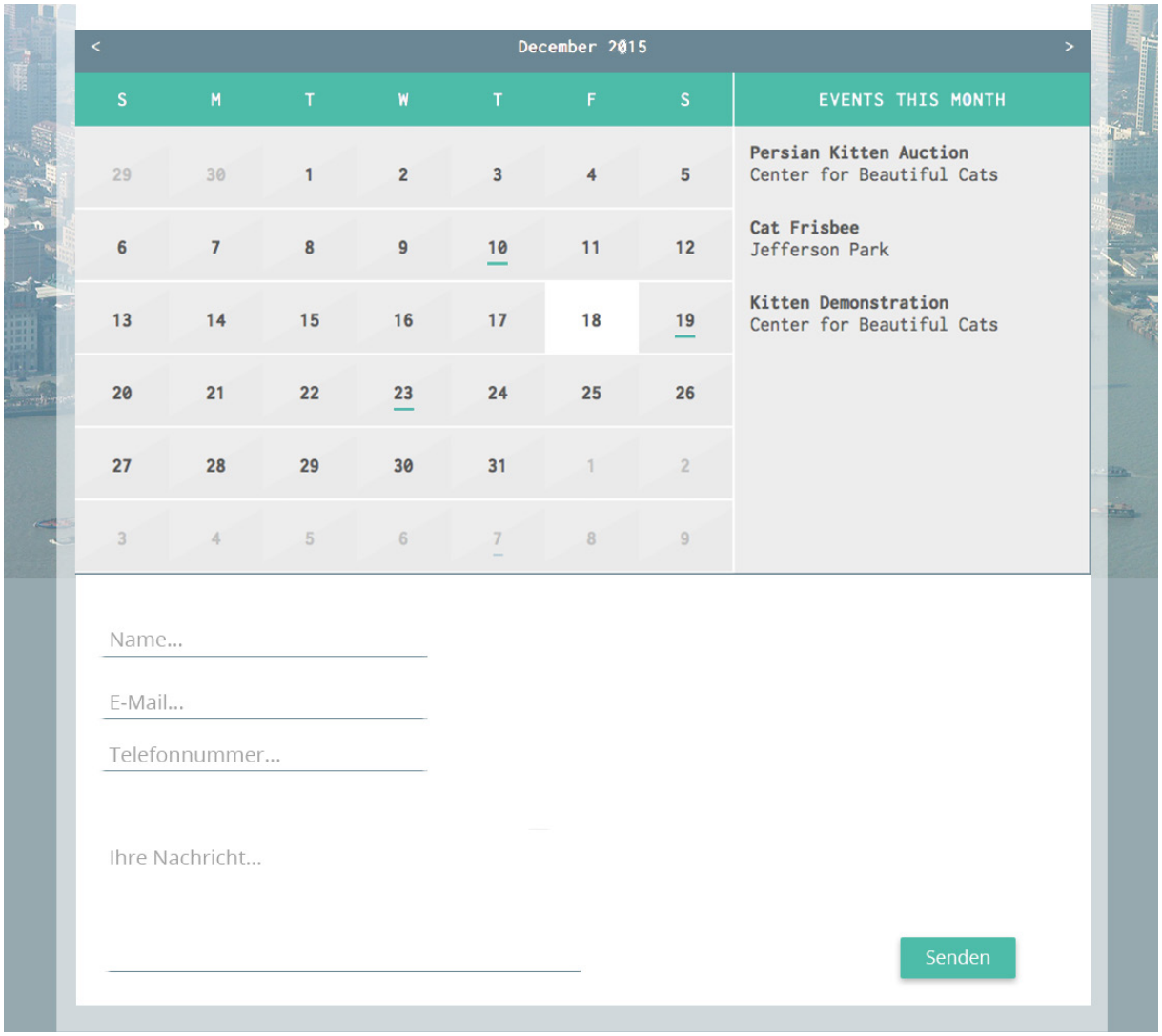


Abbildung 5.14: Modul: Kalender (oben)
Modul: Formular (unten)

Im Karten-Modul (Abbildung 5.15) können mehrere Orte durch Marker markiert und jeweilige Informationen angegeben werden. In diesem Fall stehen die Marker für Standorte des Unternehmens. Durch einen Klick auf einen Marker werden die Informationen zu diesem Standort angezeigt.

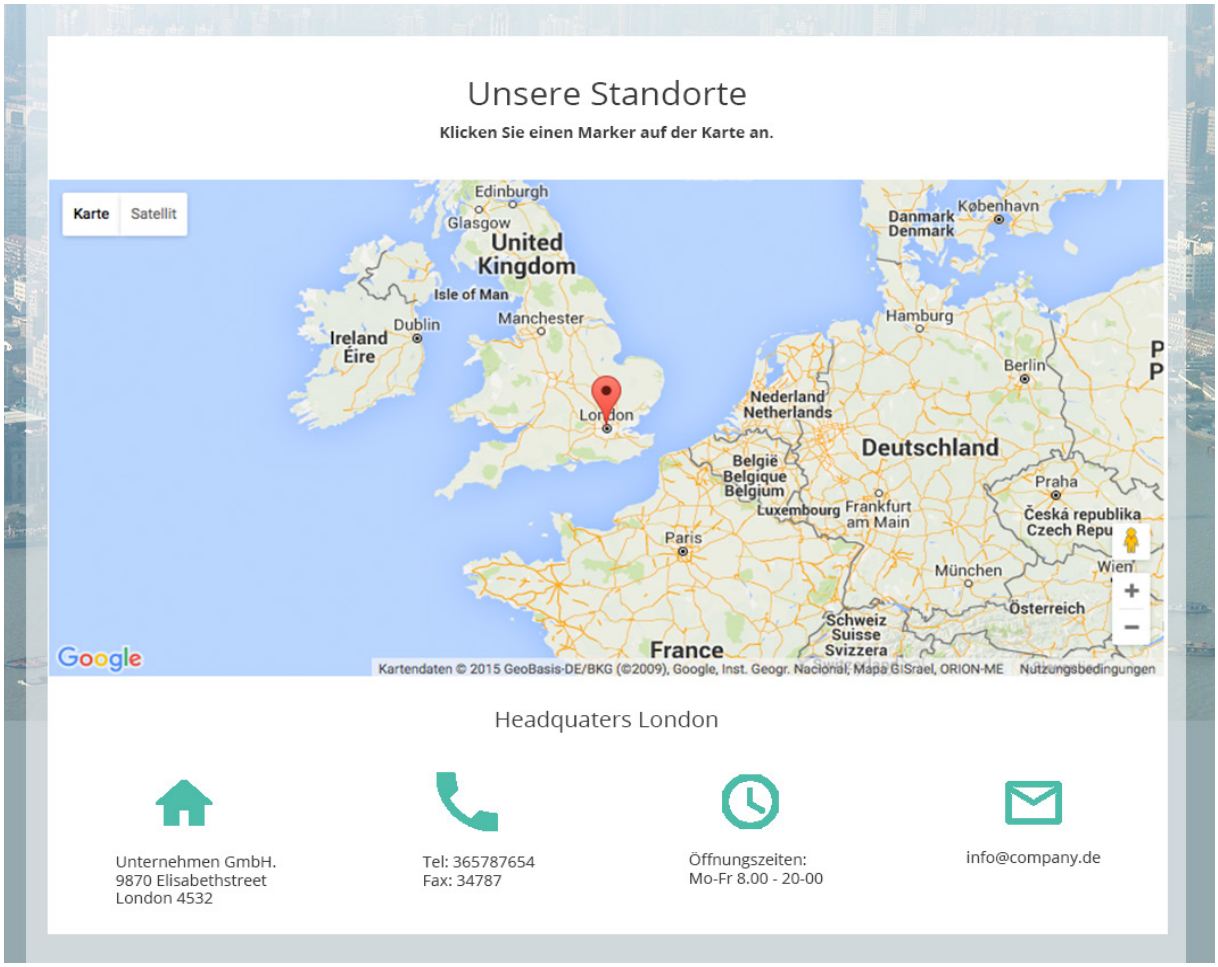


Abbildung 5.15: Modul: Karte

In Abbildung 5.16 wird ein einfacher Artikel dargestellt. Die Textbreite beschränkt sich auf eine maximale Breite, um die Lesbarkeit zu gewährleisten.

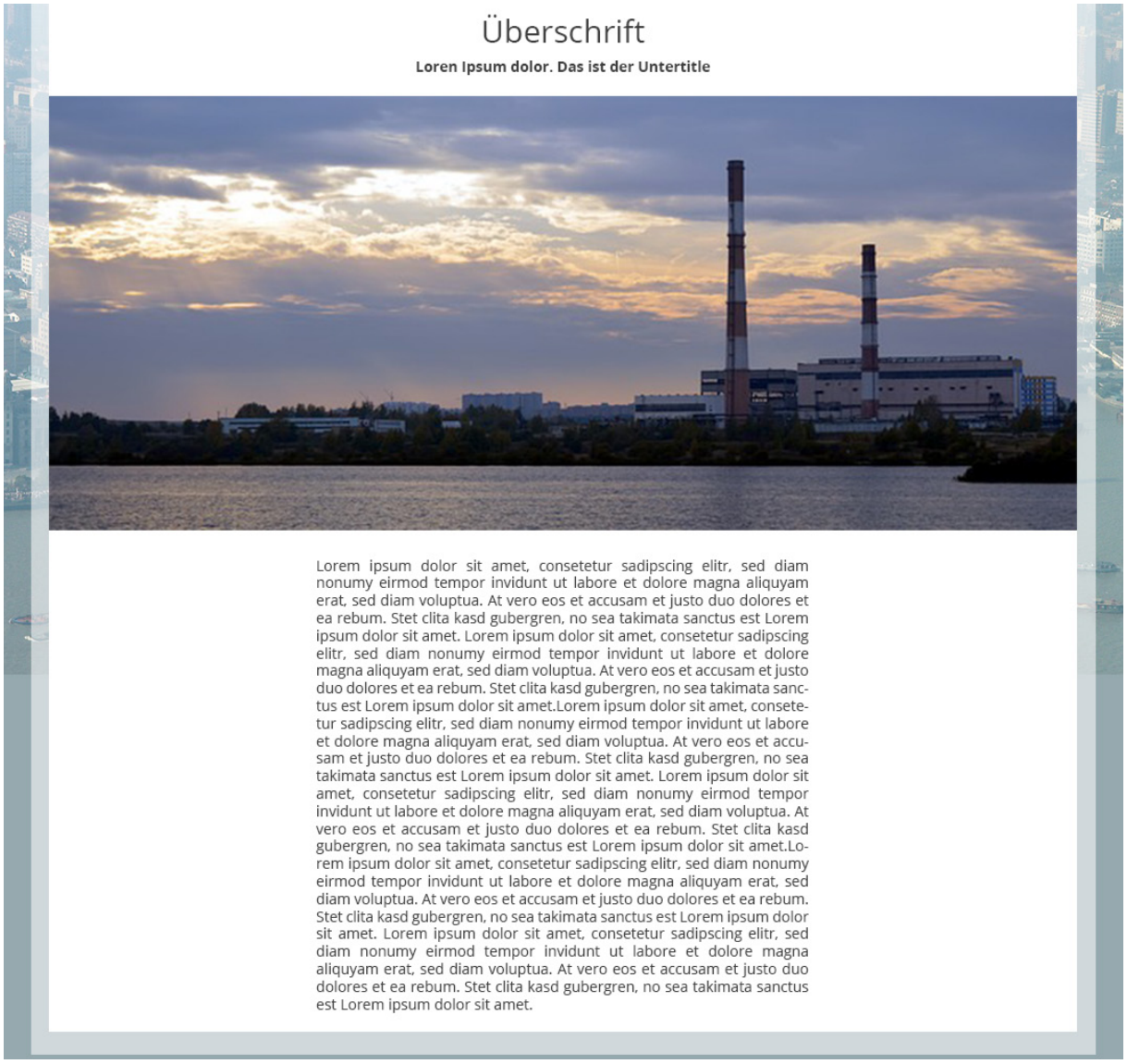


Abbildung 5.16: Modul: Artikel

Je nach Kundenwunsch können die Module unterschiedlich auf den Seiten kombiniert werden.

6 Umsetzung des Frontend-Templates

Die nächste Projektphase ist erreicht. Nun werden die in dem vorherigen Kapitel erstellten Photoshop-Designs mit den im Kapitel 3 erklärten Frontend-Techniken technisch umgesetzt. Dabei wird nach dem Atomic-Design-Prinzip vorgegangen. Die Erfahrung lehrt, dass die Konzeption eines Webprojektes selten 1:1 umgesetzt werden kann, da während der Entwicklung Schwierigkeiten oder effektivere Möglichkeiten aufkommen, die bei der Erarbeitung des Design nicht berücksichtigt wurden.

6.1 Atomic Design

2013 stellte der Frontend-Designer Brad Frost in seinem Blog (*bradfrost.com*) seinen Ansatz für das Design und die Entwicklung vor, genannt Atomic Design. Er suchte sich dafür die Inspiration in der Chemie, in der alles aus Organismen, die wiederum aus Molekülen und diese aus Atomen bestehen. Und genauso kann, laut Brad Frost, auch eine Webseite beschrieben werden. Dieser Ansatz soll nicht vorgeben, wie eine Webseite auszusehen hat, sondern wie sie aufgebaut werden kann. Nach diesem Prinzip soll das Template dieser Arbeit entstehen.

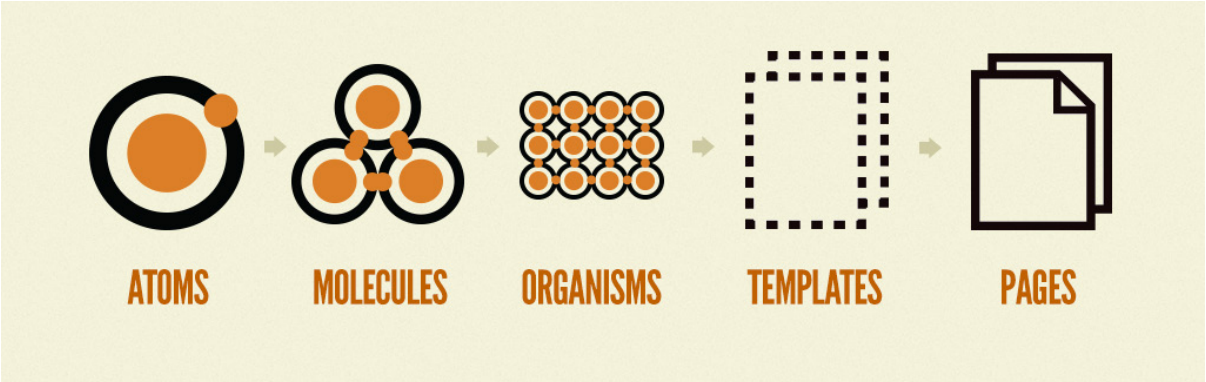


Abbildung 6.1: Bildliche Darstellung von Atomic Design (*bradfrost.com*)

Die Atome von Webseiten sind die kleinsten Elemente, also HTML-Tags wie Überschriften, Buttons oder Eingabefelder. Atome können nicht mehr weiter geteilt werden ohne ihren Sinn zu verlieren. Sie werden zu Molekülen kombiniert, die einem höheren Zweck dienen. Zum Beispiel die Atome Überschrift, Text-Absatz und Bild sind einzeln eher ohne Bedeutung, zusammen jedoch ergeben sie einen Artikel. Das Gute an Atomen und Molekülen ist die Wiederverwendbarkeit. Sie verbinden sich auf verschiedene Art und Weise zu unterschiedlichen Zwecken zusammen.

Der nächste Schritt ist das Kombinieren von Molekülen zu Organismen, die relativ konkrete Blöcke einer Webseite darstellen. Organismen können aus verschiedenen Molekülen zusammengesetzt sein, wie ein Header aus einem Logo (wobei ein Logo noch unter Atome zählt) und einer Navigation oder aus mehreren gleichen Molekülen, wie der Inhaltsbereich der Startseite eines Blogs aus mehreren Artikelvorschauen besteht. Diese Verbindungen stellen nun Komponenten dar, die für sich stehen und beliebig auf einer Webseite platziert werden können.

Um nun die Analogie der Chemie zu verlassen, kehrt Brad Frost zu mehr webtechnischen Begriffen zurück. Die nächste Ebene seines Atomic Designs sind Templates. Templates werden als Demo-Seiten verwendet, um Kunden „Etwas“ zu präsentieren. Sie beinhalten die Organismen und geben an, wo sie platziert werden. Bisher stehen in den Demo-Seiten oder Templates nur „Lorem Ipsum“ und Platzhalterbilder. Um die letzte Stufe des Atomic Design zu erreichen, wird konkreter Content in die Seite eingefügt. (Frost, 2015)

Die Einteilung in die Komponenten von Frosts Ansatz ist subjektiv und kann für jedes Template von jedem Entwickler anders umgesetzt werden.

6.2 Einrichtung des Projektes

Ordnerstruktur

Begonnen wird mit der Erstellung eines Projektordners und der Einrichtung von Grunt (Task Runner), Assemble (Static Site Generator) und weiteren nützlichen Tools. Ganz im Sinne eines Task Runners, der einem Entwickler

unliebsame Aufgaben abnimmt, gibt es auch für die Projekteinrichtung ein nützliches Plugin aus der Assemble-Familie. Der Yeoman Generator von dem Github User hariadi (*github.com*) erstellt in wenigen Sekunden eine Assemble-Boilerplate mit allen benötigten Ordner, einer *package.json*-Datei, ein eingerichteter Task Runner (Grunt) und vielem mehr.

Um dieses Plugin über die Kommandozeile nutzen zu können, wurde es zunächst global mit Hilfe des Node Package Managers (npm) installiert:

```
npm i -g generator-assemble
```

In der Kommandozeile wird in den neu erstellten Ordner „fe_template“ mit dem Befehl *cd* navigiert und der Befehl

```
yo assemble
```

ausgeführt. Anschließend müssen noch ein paar projektbezogene Fragen beantwortet werden und der Generator erledigt seine Arbeit. Nach einigen Anpassung der Boilerplate steht folgende Ordnerstruktur:

```
- node_modules
- src
      - assets
            - less
            - fonts
            - img
            - js
            - vendor
      - data
      - templates
            - layouts
            - pages
```

```
            - partials
- package.json
- Gruntfile.js
```

Quellcode 6.1: Ordnerstruktur zu Projektbeginn

Für Aufgaben die Grunt übernehmen soll, müssen Plugins installiert werden. Alle dafür benötigten Ordner und Dateien finden sich in **node_modules** wieder.

Im Ordner **src/assets** liegen Dateien, die für das Template nötig sind wie Less-, JavaScript- oder Bilddateien. Externe CSS- und JavaScript-Plugins werden in den Ordner **vendor** gelegt. In **fonts** kommen die Font-Dateien, die die Icons als Webfont enthalten.

Bei Assemble-Projekten wird der Content, auch wenn es sich nur um Fülltext und Beispielbilder handelt, vom HTML-Code separiert und im JSON-Format im Ordner **data** abgespeichert.

Das Markup des Templates liegt in Form von Handlebars-Dateien (**.hbs*) in **templates**. Der Unterordner **layouts** enthält eine oder mehrere Dateien, welche die Struktur von den Template-Seiten vorgeben und Komponenten enhalten, die auf jeder Unterseite vorkommen. Für jede Unterseite wird eine hbs-Datei im Ordner **pages** erstellt, in die jeweils die Moleküle und Organismen eingebunden werden. Und genau diese liegen neben den Atomen im Ordner **partials**.

Die Datei **package.json** enthält projektbezogene Meta-Daten wie Name, Version, Beschreibung und Grunt-Plugins.

Die Routineaufgaben die der Task Runner Grunt übernehmen soll, werden in der **Gruntfile.js** konfiguriert.

Quellcode 6.2:
Ausschnitt *Gruntfile.js*

```
17
18 module.exports = function(grunt) {
19
      …
23    // Project configuration.
24    grunt.initConfig({
        …
162   });
      …
185 };
```

24 Die Aufgaben werden in die Funktion *gunt.initConfig();* geschrieben.

Grunt ist ein Konsolen-Programm und die einzelnen Aufgaben können direkt über den Befehl *grunt TASKNAME* ausgeführt werden. Durch *yo assemble* wurden schon einige nützliche Plugins vorinstalliert.

Das Herzstück der Static Site Generator-Template-Entwicklung ist der *assemble*-Task (Quellcode 6.3). Wird er durch Grunt ausgeführt, fügen sich alle Code-Fragmente zu statischen HTML-Seiten zusammen und werden gemeinsam mit dem Ordner *assets* in einen neuen Ordner „dist“ abgespeichert. Der neue Ordner liegt auf der selben Ebene wie der *src*-Ordner und enthält alle Dateien, die für die Webseite nötig sind, um sie auf einen Server zu laden.

```
80      assemble: {
81        pages: {
82          options: {
83            flatten: true,
84            assets: '<%= config.dist %>/assets',
85            layout: '<%= config.src %>/templates/layouts/default.hbs',
86            data: '<%= config.src %>/data/*.{json,yml}',
87            partials: '<%= config.src %>/templates/partials/*.hbs'
88          },
89          files: {
90            '<%= config.dist %>/': ['<%= config.src %>/templates/pages/*.
hbs']
91          }
92        }
93      },
```

Quellcode 6.3: Ausschnitt *Gruntfile.js,* Konfiguration des *assemble*-Tasks

82 In den Optionen wird definiert, wo sich welche Dateien befinden, die Assemble benötigt. Die Angabe „flatten: true," bewirkt eine Kürzung der Dateipfade im Ordner *dist*. Die Ausdrücke „<%= ... %>" sind festgelegte Variablen.

89 Für jede *.hbs*-Datei im Ordner *pages* wird eine statische HTML-Datei in *dist* generiert.

Um das Template unter nahezu realen Bedingungen zu testen, wird der Task *connect* angelegt (Quellcode 6.4). Er startet einen lokalen Server, der unter der Adresse „localhost: 9090“ im Browser zu erreichen ist.

```
3     connect: {
64       options: {
65         port: 9090,
66         livereload: 35729,
67         // change this to '0.0.0.0' to access the server from outside
68         hostname: 'localhost'
69       },
70       livereload: {
71         options: {
72           open: true,
73           base: [
74             '<%= config.dist %>'
75           ]
76         }
77       }
78     },
```

Quellcode 6.4:
Ausschnitt *Gruntfile.js,* Konfiguration des *connect*-Tasks

64 Die Adresse des lokalen Servers.

70 Beim Ausführen von connect wird durch die Option „open: true“ automatisch ein Browserfenster oder -tab mit der Serveradresse geöffnet. Als Standart-ziel-Ordner (base) wird der *dist*-Ordner definiert.

Ein Plugin ermöglicht das automatische Ausführen von Tasks. Der auf Änderungen in festgelegten Dateien reagierende Listener nennt sich *watch* (Quellcode 6.5).

```
31 watch: {
32  assemble: {
33   files: ['<%= config.src%>/{data,templates}/{,*/}*.{md,hbs,yml,json}'],
34   tasks: ['assemble']
35  },
36  livereload: {
37   options: {
38    livereload: '<%= connect.options.livereload %>'
39   },
40   files: [
41    '<%= config.dist %>/{,*/}*.html',
42    '<%= config.dist %>/assets/{,*/}*.css',
43    '<%= config.dist %>/assets/{,*/}*.js',
44    '<%= config.dist %>/assets/{,*/}*.{png,jpg,jpeg,gif,webp,svg}'
45   ]
46  },
…
61 },
```

Quellcode 6.5:
Ausschnitt *Gruntfile.js,* Konfiguration des *watch*-Tasks

33, 34 Sobald es eine Änderung an einer Datei im *data*- oder *template*-Ordner gibt, soll einmal *assemble* ausgeführt und somit die statischen HTML-Seiten in *dist* nochmal neu generiert werden.

36 Werden Dateien im *dist*-Ordner aktualisiert, z.B. duch *assemble*, wird ein Reload der Browserfenster ausgelöst.

Der *watch*-Task kann je nach Bedarf beliebig erweitert werden. Zusätzlich zu den bereits installierten Plugins wird in diesem Fall noch ein weiteres zum Kompilieren von Less- zu CSS-Code benötigt. Der *less*-Task lässt sich über npm von github installieren (*github.com*) und in der *Gruntfile.js* konfigurieren (Quellcode 6.6).

```
133     less: {
134       development: {
135         options: {
136           compress: true
137         },
138         files: {
139           '<%= config.dist %>/assets/css/style.css': '<%= config.src %>/
assets/less/styles.less',
140         }
141       }
142     },
```

Quellcode 6.6:
Ausschnitt *Gruntfile.js*, Konfiguration des *less*-Tasks

138 Die Datei *style.less* soll zu *style.css* umgewandelt werden.

136 Beim Kompilieren sollen unnötige Leerzeichen und Absätze entfernt werden (Minifizieren), um Speicherplatz und Ladezeit zu sparen.

Das Template sieht eine Bildergalerie mit Thumbnails vor. Damit nur die Originalbilder dem Projektordner hinzugefügt werden müssen und nicht zusätzlich noch das entsprechende Thumbnail, wird das Plugin *resize_crop* installiert (*npmjs.com*) und konfiguriert (Quellcode 6.7).

Quellcode 6.7:
Ausschnitt *Gruntfile.js*, Konfiguration des *resize_crop*-Tasks

```
143     resize_crop: {
144       image_group: {
145         options: {
146           format: "jpg",
147           gravity: "center",
148           height: 100,
149           width: 100
150         },
151         files: {
152           '<%= config.src %>/assets/img/gallery/thumbnails': [
153             '<%= config.src %>/assets/img/gallery/*.jpg'
154           ],
155         }
156       }
157     },
```

145 Bilder im jpeg-Format sollen weitestgehend auf 100px mal 100px runter skaliert und dann um den Mittelpunkt genau auf diese Maße zu geschnitten werden.

151 Bilder aus dem Ordner *src/.../gallery* sollen als Thumbnails in *dist/.../thumbnails* kopiert werden.

Normalerweise bedarf es zur Aktivierung der Plugins für jedes die Funktion:

```
grunt.loadNpmTask('PLUGINNAME');
```

Mit Hilfe von *load-grunt-tasks* ist nur noch eine Zeile Code nötig, um alle installierten Plugins zu aktivieren:

```
20
21  require('load-grunt-tasks')(grunt);
22
```

Quellcode 6.8: Ausschnitt *Gruntfile.js,* Einbindung des Plugins *load-grunt-tasks*

Es bedient sich dabei an der Liste installierter Plugins aus der Datei *package.json*.

Weitere Tasks, die Grunt ausführen soll, sind *copy*, wodurch die Dateien aus *src/assets/* in *dist/assets/* kopiert werden und *clean*, wodurch alte Dateien aus dem *dist*-Ordner gelöscht werden, bevor die neu kompilierten Versionen hineingeschrieben werden.

Grundsätzlich können die einzelnen Tasks durch die Eingabe von

```
grunt TASKNAME
```

in die Kommandozeile ausgeführt werden. Grunt ermöglicht es, mehrere Aufgaben unter einem gemeinsamen Namen zusammen ausführen zu lassen. Hier wurden drei solche Aufgabengruppen über die Funktion *grunt.registerTask();* definiert:

```
166  grunt.registerTask('server', [
167    'build',
168    'connect:livereload',
169    'watch'
170  ]);
171
172  grunt.registerTask('build', [
173    'clean',
174    'resize_crop',
175    'less',
176    'copy',
177    'assemble'
178  ]);
179
180  grunt.registerTask('default', [
181    'build'
182  ]);
```

Quellcode 6.9:
Ausschnitt *Gruntfile.js,* Anlegen von Aufgaben-Gruppen

166 Wird mit der eigentlichen Programmierarbeit begonnen, wird der Befehl *grunt server* in die Konsole eingegeben. „grunt server" stößt die Aufgaben in der Gruppe *server* an. Die erste Aufgabe, *build*, löst die ab Codezeile 172 aufgelisteten Einzeltasks aus. Durch *connect:livereload* wird ein neues Browserfenster geöffnet, das die *index.html*-Datei aus dem *dist*-Ordner anzeigt. Der letzte Tasks, *watch*,

wird von nun an dauerhaft ausgeführt. Er horcht auf Änderungen in vorab definierten Ordnern und Dateien, um dann den entsprechenden Befehl und einen Reload des Browserfensters anzuführen.

```
180
```

Die Aufgabengruppe *default* wird bei der Konsoleneingabe von *grunt* ausgelöst.

Die Konfiguration von Grunt ist damit vorerst beendet. Es ist natürlich nicht ausgeschlossen, dass sich während der Entwicklung noch weitere Aufgaben ergeben, die sich durch den Task Runner automatisieren lassen und jederzeit nachgetragen werden können.

Nicht nur die *Gruntfile.js* ist projektübergreifend gleich. Auch andere Dateien tauchen, bei gleichem Workflow, in jedem Projekt auf.

assets/less/variables.less: In der *variables.less*-Datei werden Less-Variablen und -Mixins gespeichert, die sehr häufig im Projekt auftauchen wie Farben, Schatten, Abstände etc. Zu diesem Zeitpunkt werden allerdings vorerst nur vier Breakpoint-Variablen für die responsive Darstellung definiert, *@screen-xs* bei 480px, *@screen-sm* bei 768px, *@screen-md* bei 992px und *@screen-lg* bei 1200px.

assets/less/style.less: In diese Datei werden alle übrigen Less-Dateien über *@import*-Befehle eingebunden. Die erste Datei kann schon mit *@import „variables.less";* eingetragen werden.

assets/js/init.js und functions.js: Eigener JavaScript-Code wird objektorientiert in *functions.js* geschrieben. Die *init.js* initalisiert bei Bedarf die einzelnen Objekte. Als Beispiel

dient die Bildergalerie, für die in der *funtions.js* das Objekt „gallery“ mit entsprechenden Methoden und Variablen angelegt wird. Die *init.js* prüft bei jedem Seitenaufruf, ob die Galerie auf der Seite enthalten ist, um in diesem Fall das JavaScript-Objekt *gallery* aufzurufen.

assets/vendor/normalize.css: Die verschiedenen Browser setzen einige CSS-Eigenschaften mit minimalen Variationen um. Zum Anpassen dieser Unterschiede wird *normalize.css* verwendet (*https://necolas.github.io/normalize.css/*).

assets/vendor/modernizr-custom.js: Einige moderne CSS3-Eigenschaften können durch manche Browserversionen nicht interpretiert werden. Modernizr (*modernizr.com*) ist ein Feature-Detector, welcher u.a. ermittelt, ob der genutzte Browser bestimmte Eigenschaften umsetzen kann oder nicht und dementsprechend in den *html*-Tag CSS-Klassen einträgt, die auf die Kompatibilität oder Inkompabilität hinweisen. Für Fälle bei denen eine Browserversion die Techniken nicht korrekt umsetzen kann, werden alternative Stylings in eine extra Less-Datei geschrieben.

templates/partials/m_bottom_script.hbs: Zum Einbinden der bisher angelegten JavaScript-Dateien und jQuery wird diese Datei angelegt.

templates/partials/m_head.hbs: Auch der *head*-Tag wird in jedem Webprojekt benötigt und in die Datei *m_head.hbs* geschrieben (Quellcode 6.10).

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="description" content="{{metadescription}}">
<title>{{title}} | {{site.title}}</title>

<link href="{{assets}}/vendor/normalize.css" rel="stylesheet">
<link href="{{assets}}/css/style.css" rel="stylesheet">
```

Quellcode 6.10: *m_head.hbs*, enthält die Inhalte des *head*-Tags

5 Für ganz allgemeingültige Daten für die gesamte Website wird eine *.yml*-Datei names „site“ angelegt. Bisher enthält sie nur den Webseiten-Titel (*title*).

7, 8 Die bisher angelegten CSS-Dateien werden eingebunden.

templates/layouts/default.hbs: Das HTML-Grundgerüst wird in die *default.hbs* geschrieben (Quellcode 6.11).

```
<!DOCTYPE html>
<html lang="de">
  <head>
    {{> m_head}}
  </head>
  <body>
    {{> body}}
    {{> m_bottomScript}}
  </body>
</html>
```

Quellcode 6.11: *default.hbs,* enthält das HTML-Grundgerüst des Templates

5, 9 Durch Handlebars-Ausdrücke wird angegeben, dass die zuvor angelegten Dateien an diesen Stellen druch Assemble eingefügt werden sollen.

8 Der Ausdruck „body“ ist von Handlebars für die Inhalte der *.hbs*-Dateien aus *pages* reserviert.

templates/pages/index.hbs: Als letztes wird noch eine erste Unterseite, die Startseite, angelegt, die bis auf den Seitentitel und die Seitenbeschreibung im YFM-Format noch nichts enthält (Quellcode 6.12).

Quellcode 6.12: *index.hbs,* für die Inhalte der Startseite. Enthält bisher nur Meta-Informationen

```
1 ---
2 title: Home
3 metadescription: Seitenbeschreibung
4 ---
```

Hiermit ist die Projekteinrichtung abgeschlossen. Die Ordner und ersten Dateien sind angelegt und der Task Runner Grunt weiß welche Aufgaben er wann zu erledigen hat. Um die bisherige Arbeit in zukünftigen Projekten nutzen zu können, wird von dem aktuellen Projektstand eine Kopie als Vorlage abgespeichert.

6.3 Umsetzung von Layout und Modulen

Das Projekt ist eingerichtet, daher folgt die Erstellung der Module und des Layouts nach dem Prinzip des bereits erklärten Atomic Designs. Bevor mit der täglichen Programmierarbeit begonnen wird, muss der Task Runner gestartet werden. Dafür wird *start grunt server* in die Kommandozeile eingegeben. Der Ausdruck „start“ führt den dahinter folgenden Befehl in einem neuen Konsolenfenster aus. Um neben der Entwicklung die Ergebnisse direkt zu testen, werden noch die Browser Mozilla Firefox, Google Chrome und der Internet Explorer geöffnet.

Zu Beginn wird die Datei *variables.less* um einige projektbezogene Angaben wie Farben und dem mehrmals vorkommenden Schlagschatten, erweitert (Quellcode 6.13).

```
47 @main-color:                 #547C85;
48 @highlight-color:            #00B99E;
49 @font-color:                 #505050;
50
51 .shadow-1deep () {
52    box-shadow: 0 2px 2px 0 rgba(0, 0, 0, 0.14), 0 3px 1px -2px rgba(0,
0, 0, 0.2), 0 1px 5px 0 rgba(0, 0, 0, 0.12);
53 }
```

Quellcode 6.13: *variables.less*, enthält Variablen mit häufig eingesetzten Werten

Eine weitere Less-Datei, „glyphicon.less“ wird benötigt, um die Icon-Font einzubinden und den im Template verwendeten Icons CSS-Klassen zu zuweisen. Diese Klassen können dann in HTML-Tags angewandt werden. Auf der Webseite *icomoon.com/app* wurde eine Icon-Font mit den relevanten Icons erstellt. Diese Font-Dateien werden in den Ordner *asstes/fonts* geladen und in der *glyphicon.less*-Datei eingebunden.

Atome

Die Dateien, die Markup für Atome enthalten, bekommen der Übersichtlichkeit halber als Präfix ein „a_“.

Zu der Gruppe der Atome zählen die Überschriften (Abbildungen 6.2 - 6.4). Im Template-Design befinden sich drei verschiedene Typen plus zwei Untertitel-Arten.

Abbildung 6.2: Modul „Große Überschrift“: Typ 1

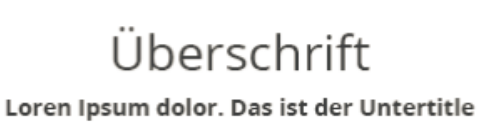


Abbildung 6.3:
Überschriften-Typ 2 mit Unterüberschriften-Typ 1

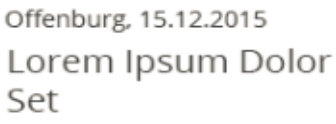


Abbildung 6.4:
Überschriften-Typ 3 mit Unterüberschriften-Typ 2

Die Überschriften und Untertitel werden über *h1*- bis *h3*- und *p*-Tags in das Template eingebunden. Je nach Typ bekommen sie eine CSS-Klasse zugewiesen, die in der neuen Less-Datei *atoms* deklariert werden.

```
/* --- Atome --- */

/* -- Headlines -- */
.h1,.h2,.h3 {
     text-align: center;
     font-weight: 400;
     margin-bottom: 0;
}
.h1 {
     font-size: 36px;
     font-weight: 300;
```

```
12 }
13 .h2 {
14      font-size: 24px;
15 }
16 .h3 {
17      font-size: 18px;
18 }
19 .subtitle {
20      font-weight: 300;
21      text-align: center;
22      margin: 7px 0 0 0;
23 }
24 .subtitle-bold {
25      .subtitle;
26      font-weight: 600;
27 }
```

Quellcode 6.14: Ausschnitt *atoms.less*, Styling der Überschriften-Typen

4 Alle Überschriften und Untertitel sind durch „text-align: center;“ horizontal zentriert und ohne Abstand nach unten. Der auf null gesetzte Abstand rührt daher, dass es auch Module geben kann, denen keine Überschrift vergeben wird, um dennoch Platz nach oben zu haben, bekommen die Module den Abstand.

9–18 Je nach Typ variiert die Schriftgröße („font-size“), zusätzlich wird die *h1*-Überschrift mit einer geringeren Schriftdicke („font-weight“) angezeigt.

24 Der dicke Untertitel bekommt mit „.subtitle;“ alle Eigentschaften, die die Klasse *subtitle* (Zeile 19) hat, nur, dass es noch die Schriftdicke mit 600 überschreibt.

Abbildung 6.5:
runder Button

Abbildung 6.6:
Default-Button

Neben den Überschriften bilden zwei verschiedene Buttons Atome, ein runder (Abbildung 6.5) und ein eckiger (Abbildung 6.6).

```
49 /* -- buttons -- */
50
51 .btn {
52     border: none;
53     cursor: pointer;
54     background: @highlight-color;
55     color: #fff;
56     border-radius: 3px;
57     padding: 5px 13px;
58     text-decoration: none;
59     margin-top: 10px;
60     .shadow-1deep;
61 }
62 .btn-round {
63     .btn;
64     border-radius: 14px;
65     padding: 7px;
66 }
67
```

Quellcode 6.15:
Ausschnitt *atoms.less*,
Styling der Button-Typen

51 In der Klasse *btn* wird die Less-Variable *@highlight-color* verwendet, welche den Buttons die Kontrastfarbe als Hintergrundfarbe vergibt.

63 Die runde Version der Buttons erhält hier alle Eigenschaften, die auch die *btn*-Klasse hat und darunter die individuellen Eigenschaften für eine runde Darstellung.

Die Buttons können durch verschiedene HTML-Tags angezeigt werden:

```
74     <button class="btn">Ein Button</button>
75     <input type="submit" class="btn" value="Senden">
76     <span class="btn-round icon-chevron-right"></span>
```

Quellcode 6.16:
Mögliche HTML-Tags für die Button-Typen

Neben diesen zwei Arten zählen die Menü-Items noch zu der Gruppe der Button-Atome. Als dritte Atom-Gruppe werden zwei Arten von Texten im Template-Design erkannt (Quellcode 6.17). Einen Fließtext, z.B. im Artikel oder Accordion, der eine von seinem Elternelement abhängige Breite hat und einem zentrierten Fließtext, z.B. in einer Icon-Card, der über die volle Breite des Elternelementes geht.

```
110 /* -- Fließtext -- */
111 .flowing-text {
112     margin: auto;
113     padding: 20px 10px;
114     @media (min-width: @screen-sm) {
115             width: 60%;
116     }
117 }
118 .flowing-text-center {
119     text-align: center;
120 }
```

Quellcode 6.17: Ausschnit *atoms.less*, Styling der Text-Typen

114 Der weitläufige Fließtext bekommt ab dem Breakpoint *@screen-sm* (768px) eine Breite von 60% seines Elternelements.

119 Die zweite Textart wird zentriert.

Um den Rahmen der Thesis nicht zu sprengen, werden folgende Atome nur noch namentlich erwähnt und nicht ausführlicher erklärt. Im Template-Design befinden sich verschiedene Bild-Atome, ein hohes und ein niedriges Bild über die ganze Breite des Elternelements und große und kleine Thumbnails sowie eine Art von Eingabefeldern, ein Zitat und vier Arten von Icons. Das Logo, der Kalender und die Karte zählen ebenfalls zu den Atomen, da sie in weitere Teile aufgesplittet ihren Sinn verlieren. Der Kalender wurde

unter zu Hilfenahme der Anleitung von *mrknowing* erstellt (*mrknowing.com*, 2013). Die Karte wird über OpenStreet-Map eingebunden und durch die JavaScript-Bibliothek leaflet (*leafletjs.com*, Lizenz: *github.com/Leaflet/Leaflet/blob/master/LICENSE*) bearbeitet.

Bisher existiert eine Sammlung von einzelnen Atomen auf der Seite. Nun werden die Atome zu Molekülen gruppiert.

Moleküle

Dateien mit dem Markup für Moleküle bekommen das Präfix „m_“ vorangesetzt.

Das wahrscheinlich typischste Molekül ist die Seitennavigation. Die Navigation dieses Templates ist ein responnisves Menü, dass sich ab einer bestimmten Bildschirmbreite zusammenklappt und über ein Icon aufrufbar ist.

Zuerst wird eine *header.json*-Datei angelegt, die Testinformationen zum Logo und zur Navigationstruktur enthält und im Ordner *data* gespeichert:

```
{
	„logo“:{
		„img“:“assets/img/logo.png“,
		“alt”:”lebase”
	},
	“navigation”:[
		{
			“title”:”Home”,
			“link”:”index”,
			“subnav”: [
				{
					“title”:”Unterpunkt 1”,
					“link”:”index”
				},
				{
					„title“:“Unterpunkt 2“,
					„link“:“index“
				}
			]
		},
		{
			„title“:“Galerie“,
			„link“:“gallery“
		}
	]
}
```

Quellcode 6.18: *header.json*, Daten für den Header

Die *.json*-Datei ist unterteilt in die Komponenten Logo, für welches der Link zum Bild und der Alternativtext zum Bild angegeben sind und die Navigation, für dessen einzelne Menüpunkte der anzuzeigende Titel, der Name der zu verlinkenden Datei und falls gegeben die Struktur einer zweiten Menü-Ebene, angegeben sind.

Nun wird eine *m_nav.hbs*-Datei angelegt (Quellcode 6.19). Diese Handlebars-Datei enthält den HTML-Code für die Navigation und Handlebars-Ausdrücke für die Inhalte aus der *header.json*-Datei.

```
1  {{ #header }}
2    <nav class="nav-collapse clearfix">
3      <span class="icon-menu icon-medium"></span>
4      <ul class="clearfix first-level">
5      {{ #each navigation }}
6        <li class="{{#is ../../basename link }}active{{/is}}navigation-
               btn navigation-btn-first-level {{#if subnav}}has-subnav{{/
               if}} ">
7          <a href="{{link}}.html" class="navigation-btn-first-level-link">
                   {{ title}}</a>
8        {{#if subnav}}
9          <ul class="second-level">
10           {{#each subnav}}
11           <li class="{{#is ../../basename link}}active{{/is}}
                  navigation-btn navigation-btn-secong-level">
12             <a href="{{ link }}.html" class="navigation-btn-second-
                  level-link"> {{ title }}</a>
13           </li>
14           {{/each}}
15         </ul>
16       {{/if}}
17     </li>
18   {{/each}}
19   </ul>
20  </nav>
21 {{/header }}
```

Quellcode 6.19: *m_nav.hbs*, enthält das Markup für die Navigation

1 Dieser Ausdruck besagt, dass folgend die Inhalte aus einer Datei namens „header“ aus dem *data*-Ordner benutzt werden sollen.

5 Der Code zwischen dem öffnenden und schließenden *each*-Ausdruck soll für jedes Element unter dem Punkt „navigation“ in der *header.json*-Datei ausgegeben werden.

9–18 Für jeden Menüpunkt wird ein *li*-Tag mit einem *a*-Tag angelegt. In Zeile 6 wird geprüft, ob der Name der aktuellen Datei aus dem *pages*-Ordner derselbe ist wie der Link des aktuellen Menüpunktes, um ihm in diesem Fall die CSS-Klasse „active“ zu vergeben. Der Ausdruck in Zeile 8 fragt ab, ob für den aktuellen Menüpunkt ein Untermenü eingetragen ist. Ist dies der Fall, wird nochmal eine neue „ul“ angelegt.

Nach zusätzlichen Stylings in der neuen Less-Datei für Moleküle *molecules* sieht das Ergebnis so aus:

Abbildung 6.7: Navigation in der Desktop-Version

Abbildung 6.8: Geschlossene Navigation in der mobilen Ansicht

Abbildung 6.9: Offene Navigation in der mobilen Ansicht

Der Menüpunkt der aktuellen Seite bekommt einen unteren Rand in der Kontrastfarbe. Menüpunkte, die mit der Maus überfahren werden, bekommen einen Hintergrund in der Kontrastfarbe. Ab der Bildschirmbreite von 765px wird die Desktopvariante angezeigt.

Als Nächstes wird die Single-Preview-Card zusammengesetzt (Quellcode 6.20). In der Version der Demo-Seite wird eine solche Karte im Gegensatz zur Konzeption für die Vorstellung eines Mitarbeiters verwendet. Durch den Klick auf den Button fährt der untere weiße Teil der Karte bis an den oberen Rand der Karte und zeigt weitere Informationen über den Mitarbeiter, in diesem Fall ein Zitat der Person.

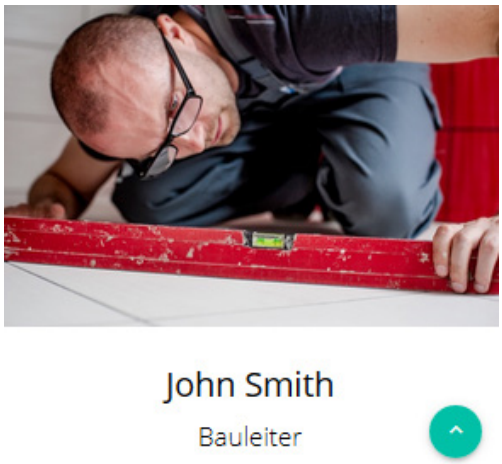


Abbildung 6.10: Modul: SinglePreviewCard, geschlossen

Abbildung 6.11: Modul: SinglePreviewCard, offen

```
 1  <div class="single-preview-card">
 2    <div class="full-width-img-small" style="background-image:url(
assets/img/mitarbeiter/man.jpg">
 3    </div>
 4    <div class="single-preview-info">
 5      <h3 class="h3">John Smith</h3>
 6      <p class="subtitle">Bauleiter</p>
 7      <blockquote class="zitat"><span>Bauen ist,…</span> </blockquote>
 8    </div>
 9    <span class="btn-round item-transition icon-expand_less"></span>
10  </div>
```

Quellcode 6.20: *m_singlepreviewcard.hbs*, enthält das Markup für eine einzelne Karte

In diesem Molekül (Quellcode 6.20) sind fünf Atome verbaut: ein kleines Bild, eine *h3*-Überschrift, ein Untertitel, ein runder Button und ein Zitat. Das Bild wird als Hintergrundbild eingefügt, da CSS hierbei die Möglichkeit hat das Bild, egal welches Format, auf dem Element flächenfüllend und zentriert zu platzieren. Für die Animation des Info-Bereiches wird JavaScript bzw. jQuery benötigt. Dafür bekommt die *function.js*-Datei ein neues Objekt.

```
35  preview = {
36    init: function() {
37      $('.single-preview-card .btn-round').on('click', function( e ) {
38        $(e.currentTarget).parent().toggleClass('active');
39      });
40    }
41  }
```

Quellcode 6.21: Ausschnitt *function.js*, enthält die Funktion zum öffnen der Karte

35 Unter „preview“ werden alle Funktionen zu diesem Modul geschrieben.

37–39 Auf den Button in der Karte wird ein Klick-Event gesetzt, das bei Auslösung dem Elternelement des geklickten Buttons die Klasse „active“ entweder gibt oder nimmt, je nachdem ob es die Klasse bereits hat oder nicht. Durch die Klasse *active* wird eine CSS-Transition (ein flüssiger Übergang von einem Eigenschaftswert zu einem Anderen) ausgelöst, für den Button, der sich um 180 Grad dreht und die Preview-Info, die einen neuen *top*-Wert von null bekommt und sich somit nach oben schiebt.

Des Weiteren finden sich in dem Template noch als Moleküle ein Artikel, ein Formular, eine einzelne Icon-Card, eine Galerie-Vorschau und ein einzelnes Accordion-Item, dessen Entwicklungen nicht genauer beschrieben wird.

Organismen

Dateien mit dem Markup für Organismen bekommen das Präfix „o_“ vorangesetzt. Als Organismen werden Komponenten bezeichnet, die eigenständig in eine Seite platziert werden.

Aus der Navigation und dem Logo wird der Header erstellt. Da dieser Organismus auf jeder Seite angezeigt wird, wird er in die Datei l*ayouts/default.hbs* geschrieben (Quellcode 6.22). Die Header-Leiste bleibt beim rauf- und runterscrollen am oberen Bildschirmrand fixiert, sodass der Nutzer an jeder Position das Menü bedienen kann.

Quellcode 6.22: Ausschnitt *default.hbs,* Einfügen von Navibation und Logo

```
16   <header>
17     {{> a_logo }}
18     {{> m_nav }}
19   </header>
```

17, 18 Mit Hilfe von Handlebars-Notation werden die Dateien *logo.hbs* und *nav.hbs* aus dem Ordner Partials in den *header*-Tag eingebunden.

Damit der Header beim Scrollen am oberen Bildschirmrand bleibt, wird JavaScript-Code benötigt (Quellcode 6.23).

```
 6 $(document).on('scroll', function() {
 7   if($(window).scrollTop()>10) {
 8     if($(window).width() >= 768) {
 9       $('header').css({
10         'position': 'fixed',
11         'top': '-10px',
12         'left': '50%',
13         'margin-left': ($('header').width()/2)*(-1)
14       });
15       $('div.site-container').css({'margin-top': nav.headerHeight+30})
16     } else {
17       $('header').css({
18         'position': 'fixed',
19         'top': '-10px',
20         'right': '-10px',
21         'width': '100%',
22       });
23       $('div.site-container').css({'margin-top': nav.headerHeight})
24     }
25   } else {
26     $('header, div.site-container').removeAttr('style');
27   }
28 });
```

Quellcode 6.23: Ausschnitt *function.js*, Einfügen der Funktionen zum fixieren des Headers beim Herunterscrollen der Webseite

5 Das HTML-Dokument bekommt ein Scroll-Event zugewiesen. Der darauffolgende Code wird beim Scrollen ausgeführt.

7 Die erste *if*-Abfrage bestimmt, ob die Navigation an den oberen Rand geheftet oder davon gelöst werden soll, ob dementsprechend CSS-Code hinzugefügt oder entfernt wird.

8 Diese *if*-Abfrage fügt je nach Navigationsversion (responsiv oder desktop) den benötigten CSS-Code hinzu.

Das Preview-Modul setzt sich zusammen aus einer beliebigen Anzahl an Single-Preview-Cards und zwei kleinen runden Buttons. Die „Karten“ werden nebeneinander in einer Reihe platziert. Diejenigen, die nicht mehr in den sichtbaren Bereichs des Templates passen, werden ausgeblendet. Der Nutzer kann durch die Buttons die Reihe nach rechts und links verschieben, um die übrigen Karten in den Viewport zu schieben.

Wie auch bei der Navigation wird als erstes eine Datei mit den Informationen in json-Fomat in den Ordner *data* geschrieben. Die Datei heißt „employees.json“ und beinhaltet Informationen zu fiktiven Mitarbeiter:

Quellcode 6.24: Ausschnitt *employees.json,* Beispielinhlate für die Preview-Karten

```
{„employees“:[
      {
            „id“:“1“,
            „name“:“John Smith“,
            „profession“:“Bauleiter“,
            „zitat“: „Bauen ist, wenn man was schafft.“,
            “img”: “building.jpg”
      },
      …
      {
            “id”:”6”,
            “name”:”Jan Supermann”,
            „profession“:“Rezeptionist“,
            „zitat“: „Rezeptionieren ist, alles im Blick zu haben.“,
            „img“: „man.jpg“
      }
]}
```

Für jeden Mitarbeiter wird eine Karte generiert (Quellcode 6.25).

```
 2 <div class="preview clearfix">
 3   <div class="single-preview-card-row item-transition">
 4     {{#employees}}
 5      {{> m_single_preview_card }}
 6     {{/employees}}
 7   </div>
 8   <span class="icon-chevron-left btn-round preview-move-btn"></span>
 9   <span class="icon-chevron-right btn-round preview-move-btn"></span>
10 </div>
```

Quellcode 6.25: o_*preview.hbs,* Markup der Preview mit Einfügen der Preview-Karten

2 Der *div*-Tag „preview“ bekommt eine Breite von 100% und eine Höhe von 300px. Zusätzlich erhält es noch die Eigenschaft „overflow: hidden“, damit alle Inhalte, in dem Fall die Karten, die über die Breite der Preview hinausgehen, nicht sichbar sind.

3 In der „single-preview-card-row“ werden die einzelnen Karten-Moleküle nebeneinander aufgereiht.

4–6 Für jeden Mitarbeiter in der *.json*-Datei wird eine Single-Preview-Card generiert. In der Datei *m_single_preview_card* wurden die bisherigen Inhalte durch Handlebars-Ausdrücke ersetzt.

8–9 Durch die Buttons wird die Reihe um je eine Karten-Breite nach links bzw. nach rechts verschoben.

Um zwei Click-Events wurde die *init*-Methode des *preview*-Objektes in der *function.js* erweitert (Quellcode 6.26).

```
54 $('.preview .icon-chevron-right').on('click', function() {
55   var currentLeft = parseInt($('.single-preview-card-row').
css('left'));
56   var wSingleMod = $('.single-preview-card').length*295;
57   var wMods = $('.preview').width();
58   if((wSingleMod-wMods)*(-1) < currentLeft) {
59     $('.single-preview-card-row').css('left', currentLeft-295+'px');
60   }
61 });
62 $('.preview .icon-chevron-left').on('click', function() {
63   var currentLeft = parseInt($('.single-preview-card-row').
css('left'));
64   if(currentLeft < 0) {
65     $('.single-preview-card-row').css('left', currentLeft+295+'px');
66   }
67 });
```

Quellcode 6.26: Ausschnitt *function.js,* Einfügen der Funktionen zum hin und her fahren der Preview-Leiste

54, 62 Jedem Button wird ein Click-Event zugewiesen, in dessen Funktionen die aktuelle Position der Single-Preview-Card-Row ermittelt wird.

58, 64 Je nachdem, ob die Reihe noch nicht das linke oder rechte Ende erreicht hat, wird sie um die Länge einer Karte verschoben.

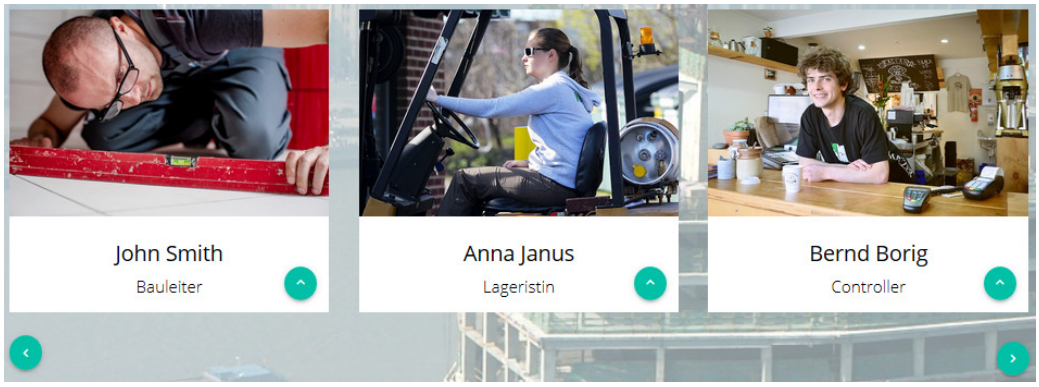


Abbildung 6.12: Preview-Module mit Beispiel-Inhalten aus der *employees.json*

Für eine letzte hier vorgestellte Implementierung eines Organismus wird das Map-Modul genutzt. In der Karte von OpenStreetMaps werden mit Hilfe der JavaScript-Bibliothek leaflet einige Marker, die Standorte markieren, platziert. Beim Anklicken der Marker werden unterhalb der Karte standortbezogene Informationen angezeigt.

Die Standortdaten werden in eine „locations.json“-Datei geschrieben und in den Ordner *data* gespeichert. Sie ist ähnlich aufgebaut wie die *employees.json* und wird daher nicht aufgeführt.

Der Organismus wird in der Datei *o_map.hbs* geschrieben:

```
 4 <div id="map"></div>
 5 <div class="clearfix map-button">
 6   <input type="button" class="btn pull-right" id="showAllMarkers"
value= "Zeige alle Marker" />
 7 </div>
 8 <div class="locationinfo-placeholder">
 9     {{#locations}}
10       <div id="{{#ident}}" class="locationinfo">
11         <h3 class="h3">{{#name}}</h3>
12         {{> m_single_icon_card_map }}
13       </div>
14     {{/locations}}
15 </div>
```

Quellcode 6.27: o_map.*hbs,* Markup des Karten-Moduls

4 Hier wird, mit Hilfe von JavaScript eine Karte eingebunden.

5–7 Ein rechtsbündiger Button, der bei Klick alle auf der Karte liegenden Marker anzeigen soll.

8 Der Platzhalter für die *locationinfo* soll einen bestimmten Bereich unter der Karte freihalten, bevor eine der Standortinformationen angezeigt wird.

ab 9 Für jede Location wird eine Überschrift und eine angepasste Form der Single-Icon-Card in einem *locationinfo*-Wrapper erstellt.

Um die Karte von OpenStreetMap nutzen zu können, müssen noch eine JavaScript- und eine CSS-Datei von *leaflet.com* in den Code eingebunden werden. In der *function.js* können nun die Funktion der leaflet-Bibliothek verwendet und die Karte angezeigt werden. Um die Marker auf der Karte zu platzieren, werden die Informationen, Standortname und Koordinaten, benötigt. Dafür wird die *locations.json* per Ajax nachgeladen. Den Markern wird ein Click-Event zugewiesen, das bewirkt, dass die zum Marker gehörigen Informationen als Icon-Cards unterhalb der Map eingeblendet werden.

Für die Standortinformationen müssen noch einige weitere Styling-Eigenschaften hinzugefügt werden. Je vier Icon-Cards pro Standort stecken in einem *div*-Container mit den CSS-Klassen „module“ und „col“. Die Klasse *module* wird weiter unten im Text näher erläutert.

```
29 /* -- icon-card-module -- */
30 .col {
31      display: flex;
32      justify-content: space-around;
33      flex-wrap: wrap;
34 }
```

Quellcode 6.28: Ausschnitt *organismen.less,* Styling der Spalten für die Standortinformationen

30, 31 Die Klasse *col* sorgt dafür, dass die Karten nebeneinander im selben Abstand stehen. CSS3 bietet eine einfache Möglichkeit, die Elemente einer Webseite anzuordnen: Flexbox („display: flex“).

32, 33 Für die gleichmäßige Verteilung der Kindelemente in jeder Reihe von *col* sorgt „justify-content: space-around“ und durch „flex-wrap: wrap“ können die Kindelemente bei zu wenig Platz in einer Reihe in eine weitere Reihe umbrechen.

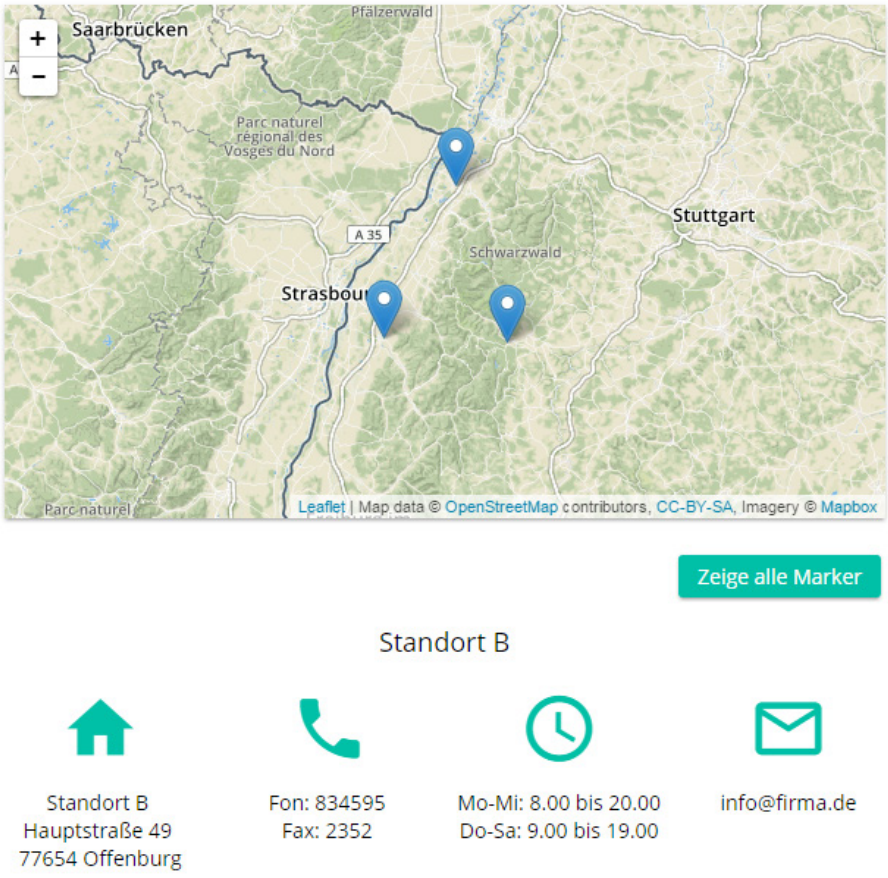


Abbildung 6.13: Karten-Modul in der Desktop-Ansicht

Neben dem Header, der Preview und dem Map-Modul bilden noch die Module Icon-Cards, Accordion und die Bildergalerie Organismen.

Template

Im Template-Teil wird das Layout realisiert, die Komponenten mit Beispielinhalten gefüllt und zu einer Demo-Seite angeordnet.

Die *default.hbs*-Datei, die bei der Projekteinrichtung angelegt wurde, wird nun um den *body*-Bereich ergänzt (Quellcode 6.29).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    {{> m_head }}
  </head>
  <body>
    <header class="site-container">
      {{> a_logo }}
      {{> m_nav }}
    </header>
    <div class="site-container" role="main">
      {{> body }}
    </div>
    <footer>
    </footer>
    {{> m_bottomScript }}
  </body>
</html>
```

Quellcode 6.29: *default.hbs,* Ausbau des Template um den *body*-Bereich

13 Dies ist ein von Handlebars festgelegter Begriff. Der Static Site Generator Assemble generiert für jede Datei im Ordner *pages* eine statische HTML-Seite mit diesem Layout, wobei er die Inhalte aus den *pages*-Dateien anstelle von *{{> body}}* einsetzt.

Für das Template wird festgelegen, dass der Hintergrund ein Bild bekommt, das jederzeit den Bildschirm ausfüllt (Quellcode 6.30).

```
5 html {
6    background: #B1C5CE url(../img/crane-1028962_1920.jpg) no-repeat
     left center;
7    background-size: cover;
8
9 }
```

Quellcode 6.30: Ausschnitt *main.less,* Angabe eines Hintergrundbildes

Den Content-Bereich grenzt seitlich die CSS-Klasse „site-container“ ab (Quellcode 6.31).

```
20 .site-container {
21      padding: 0 10px;
22      margin-top: 10px;
23      max-width: @max-content-width;
24      @media (min-width: @screen-sm) {
25              width: 90%;
26              margin: 10px auto 20px;
27              padding: 0;
28      }
29      @media (min-width: @screen-md) {
30              width: 80%;
31      }
32 }
33 div.site-container {
34      background: rgba(255,255,255, .5);
35      padding: 20px 15px 1px;
36      @media (max-width: @screen-sm) {
37              margin-left: 10px;
38              margin-right: 10px;
39      }
40 }
```

Quellcode 6.31: Ausschnitt *main.less,* Styling der seitlichen Begrenzund der Webseiten-Inhalte

24, 29 Der Container bekommt ab einer Mindestbreite des Viewports von 768px (*@screen-sm*) eine Breite von 90% und bei 992px (*@screen-md*) eine Breite von 80%.

33 Hier wird durch das *div* vor dem *.site-container* nur der Content Bereich angesprochen, nicht auch der Header, da er einen *header*-Tag hat. Der Content-Bereich bekommt verschiedene Abstände und einen transparenten, weißen Hintergrund.

Die Module werden in Reihen, entweder einzeln oder zu zweit, platziert. Die Reihen können jeweils optional einen weißen Hintergrund und eine Überschrift bekommen (Abbildung 6.14).

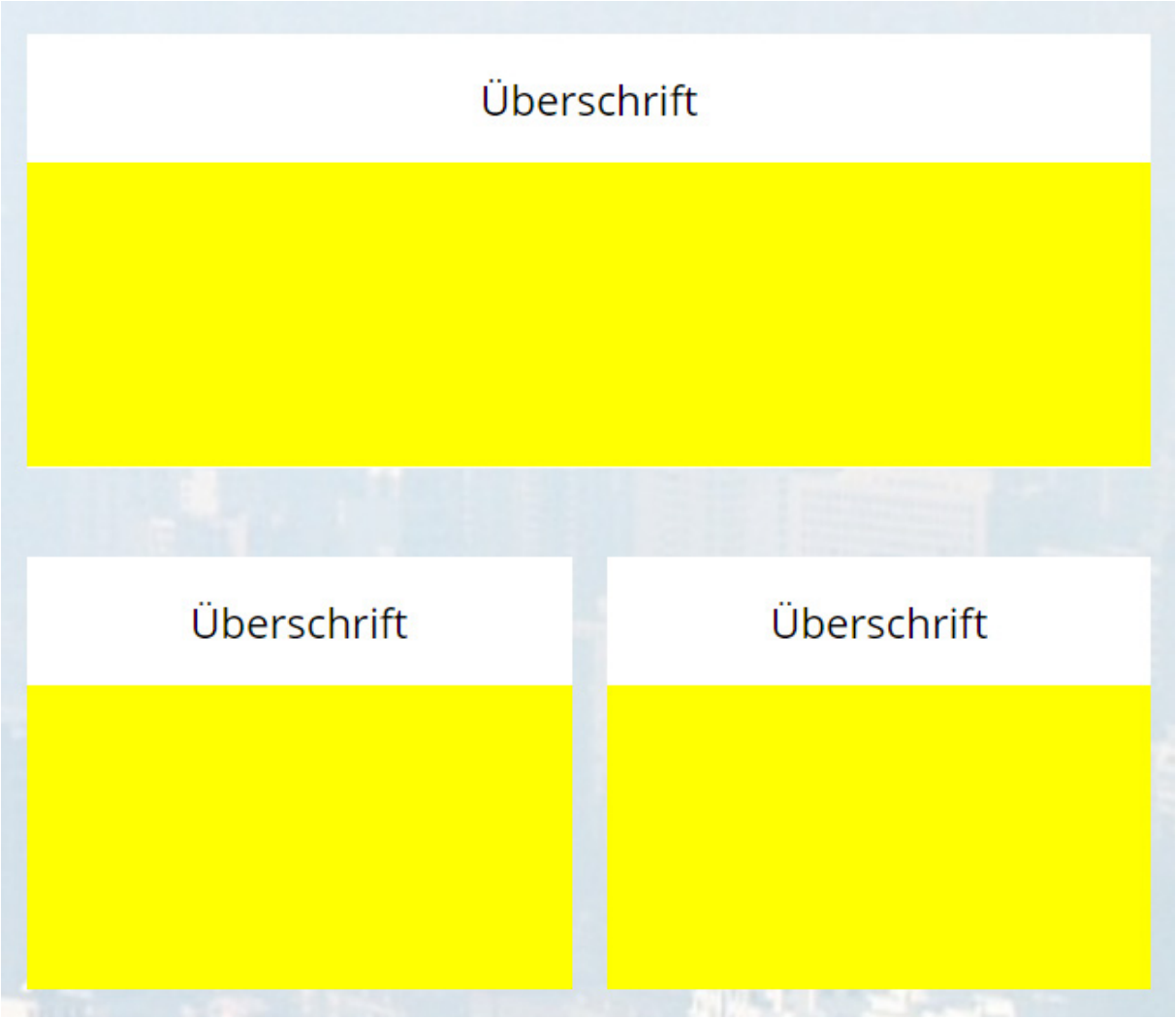


Abbildung 6.14: Anordnung der Module auf in dem Template

An Stelle der gelben Kästen kommen später die Module bzw. Organismen oder Moleküle. In HTML ist es folgendermaßen umgesetzt:

```
76 <div class="site-container" role="main">
77   <div class="one-module-row colored-bg">
78     <h2 class="h2">Überschrift</h2>
79     <div class="module">
80
81     </div>
82   </div>
83
84   <div class="two-module-row">
85     <div class="module-half-width colored-
bg">
86       <h2 class="h2">Überschrift</h2>
87       <div class="module">
88
89       </div>
90     </div>
91     <div class="module-half-width colored-
bg">    92       <h2 class="h2">Überschrift</h2>
93       <div class="module">
94
95       </div>
96     </div>
97   </div>
98 <div>
```

Quellcode 6.32: Beispielhafte Umsetzung von Reihen

77 Eine Reihe mit nur einem Module nennt sich „one-module-row“ und beinhaltet ein Modul im *div*-Tag „module“ und optional eine Überschrift und die CSS-Klasse „colored-bg“, die ihr den weißen Hintergrund verleiht.

84 Eine Reihe mit zwei Modulen heißt „two-module-row“ und enthält zusätzlich noch zwei Container namens „module-half-width“ für je eine Überschrift und ein Modul. Diese zwei Container sorgen dafür, dass Inhalte ab einer Bildschirmbreite von 768px nebeneinander stehen und bei einem schmaleren Viewport untereinander rutschen.

Bisher sind auf der Frontend-Webseite nur einzelne Elemente untereinander gestapelt. Nun werden alle Komponenten auf fünf Unterseiten in geordneter Weise in Reihen und Spalten eingefügt. Wie auch in der *layouts/default.hbs* geschieht das Einfügen über Handlebars-Ausdrücke. Jedes Modul soll vorkommen, sodass potenzielle Kunden sich einen Eindruck von den Möglichkeiten des Templates verschaffen können. Jede Unterseite wird als *.hbs*-Datei in den Ordner *pages* gespeichert.

Auf der Startseite soll eine große Überschrift, das Icon-Card- und das Preview-Modul jeweils in eine Reihe und die Galerie-Preview und das Accordion zusammen in eine Reihe eingefügt werden.

Das Ganze wird in der *index.hbs* gespeichert:

```
---
title: Home
heading: Erstklassig und weltoffen. Das Unternehmen.
---
<div class="one-module-row">
  <div class="module clearfix">
    <h1 class="h1">{{ heading }}</h1>
  </div>
</div>
<div class="one-module-row colored-bg">
  <h2 class="h2">Leistungen</h2>
  <div class="module clearfix">
    {{> o_icon_cards}}
  </div>
</div>
<div class="two-module-row">
  <div class="module-half-width colored-bg">
    <h2 class="h2">Galerie</h2>
    <div class="module clearfix">
      {{> m_gallery_preview}}
    </div>
  </div>
  <div class="module-half-width">
    <h2 class="h2">Accordion</h2>
    <div class="module clearfix colored-bg">
```

```
26        {{#accordion}}
27          {{> m_single_accordion_item}}
28        {{/accordion}}
29      </div>
30    </div>
31 </div>
32 <div class="one-module-row">
33    <h2 class="h2">Unser Team</h2>
34    <div class="module clearfix">
35      {{> o_preview}}
36    </div>
37 </div>
38
```

Quellcode 6.33: *index.hbs,* Einsetzen der verwendeten Module in die Reihen

Das Anlegen einer neuen Seite wird durch den modularen Aufbau des Templates ziemlich einfach. Für jedes Modul, das eingesetzt werden soll, wird ein neuer Zeilen-Container („one-module-row“ oder „two-module-row“) angelegt. Darin wird in den *div*-Container „module“ der Handlebars-Ausdruck, der die entsprechende Datei aus dem Ordner *partials* einfügt und optional noch eine *h2*-Überschrift eingetragen. Für den Inhalt der großen Überschrift wurde keine extra *.json*-Datei angelegt sondern in *yfm*-Format (YAML front matter) in die Zeilen 2 - 4 geschrieben und über *{{ heading }}* in Zeile 7 eingefügt.

Nach dem Anlegen von weiteren Seiten, eine Galerie sowie eine Kalendar-und-Kontakt-Seite, eine Standort-Seite und eine Artikel-Seite, ist die Frontend-Entwicklung des Templates abgeschlossen.

7 Anbindung an ein CMS

Das entstandene Frontend-Template wird nun für einen fiktiven Kunden mit einem Content Management System verbunden. Zuerst wird der Kunde mit seinen Anforderungen vorgestellt, die anschließend realisiert werden.

7.1 Der Kunde

Schmidt
Rechtsanwälte

Abbildung 7.1:
Logo des fiktiven Kunden

Die Rechtsanwaltskanzlei Schmidt möchte seinen aktuellen Internetauftritt erneuern. 1999 von Dr. jur. Thorsten Schmidt gegründet, beschäftigt die Kanzlei neben Herrn Schmidt noch drei weitere Anwälte und ist spezialisiert auf die Bereiche Wirtschaftsrecht, Arbeitsrecht, Familienrecht und Steuerrecht. Neben dem Hauptsitz in Frankfurt gibt es ein Zweitbüro in Fulda.

Anforderungen

Der Kunde möchte seine Kanzlei mit den Standorten, Mitarbeitern und Rechtsbereichen vorstellen. Dafür wurde folgende Seitenstruktur mit dem Kunden erarbeitet:

- Die Kanzlei
- Rechtsbereiche
  - Wirtschafts-/ Vertragsrecht
  - Arbeitsrecht
  - Steuerrecht
  - Familienrecht
- Standorte
- Kontakt
- Impressum

Um potenziellen Mandanten die wichtigsten Informationen direkt auf der Startseite präsentieren zu können, wird auf ihr eine große Überschrift, ein Icon-Card-Modul mit einer kurzen Vorstellung der Rechtsbereiche, ein knapper Text über die Kanzlei und das Preview-Card-Modul mit den Anwälten platziert. Auf der Unterseite *Standorte* wird das Karten-Modul eingepflegt. Unter *Rechtsbereiche* findet der Besucher einen Artikel für jeden Bereich, den die Kanzlei abdeckt. Eine Möglichkeit zur Kontaktaufnahme bietet das

Formular auf der Seite *Kontakt*.

Neben den Möglichkeiten die das Template bisher bietet, wünscht sich der Kunde noch zusätzliche Buttons auf weitere Artikel und den Ausbau des Footers. Dieser soll Kontaktinformationen zum Hauptsitz, das Copyright und den Link zum Impressum enthalten.

Da die Anwälte und Angestellten der Kanzlei alle Profis auf ihren Fachgebieten sind und daher nur wenig Erfahrung mit Webseiten-Management haben, empfiehlt sich das CMS TYPO3 Neos, da es besonders benutzerfreundlich und intuitiv ist.

Die Highlight-Farbe soll dem Blau ihres Logos entsprechen, als Hauptfarbe wird schlicht Weiß verwendet und als Hintergrundbild wird ein Bild des Bürogebäudes, in dem sich der Hauptsitz befindet, eingesetzt.

7.2 Realisierung des Projektes

Nach den Erweiterungen des Templates in Abstimmung mit dem Kunden, beginnt die Integration des Frontends in das CMS TYPO3 Neos.

7.2.1 Typo3 Neos installieren

Für die Installation des CMS wird der Anleitung auf *neos.readthedocs.org* gefolgt. Als Erstes wird im Projekteordner *htdocs* des lokalen Servers Xampp (Apache) ein neuer Ordner für die Installation angelegt. Er wird f*e_neos* genannt.

In dem Kommandozeilenfenster, das unter Windows als Administrator ausgeführt werden muss, wird zum neuen Ordner navigiert und der folgende Befehl ausgeführt:

```
php ../composer/composer.phar create-project
typo3/neos-base-distribution fe_neos
```

Dadurch werden alle für Neos benötigten Dateien heruntergeladen. Um das Projekt im Browser bequem zu erreichen, wird noch ein *virtual host* in den Apache Konfigurationen und der *hosts*-Datei des Rechners unter dem Namen „neos.demo“ eingerichtet. Xampp muss als Administrator ausgeführt und der Apache gestartet werden.

Für das Neos Setup wird im Browser nun die Adresse *neos.demo/setup* aufgerufen. Nach einem kurzen, automatischen Check der Vorraussetzungen von Flow und Neos wird ein Login-Fromular angezeigt. Das hier benötigte Passwort befindet sich in einem automatisch generierten Textdokument im Neos-Ordner. In den nächsten vier Schritten werden eine Datenbankverbindung hergestellt, ein Administrator-Account und ein neues Webseitenpaket - im Folgenden als „Package“ bezeichnet - angelegt und das erfolgreiche Setup bestätigt. Als Packagename wurde „feneos.raschmidt“ und als Seitenname „Rechtsanwalt“ gewählt. Unter *neos.demo/neos/login* kann sich nun für das Backend der Seite angemeldet werden. Das Package in dem das Frontend-Template integriert wird, liegt in dem Ordner *fe_neos/Packages/Sites/feneos.raschmidt*.

7.2.2 Integration des Frontend-Templates

Für die Integration des Frontend-Templates wird das E-Book „Neos - Schritt für Schritt“ von der Mittwald Agentur (mittwald.de) und die Dokumentation von Neos (neos.readthedocs.org) zu Hilfe genommen.

Im ersten Schritt werden die Dateien des Frontend-Templates eingefügt. Die Dateien aus dem Frontend-Projektordner *dist/assets* kommen in den Ordner *Resources/Public* des Package. Daraus ergibt sich die Ordnerstruktur in Abbildung 7.2.

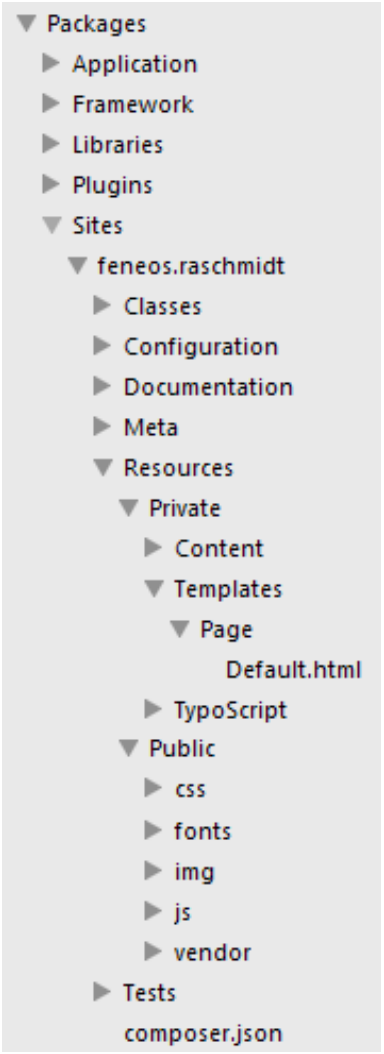


Abbildung 7.2: Ordnerstruktur in *fe_neos/Packages*

Alle Inhaltselemente wie Reihen, die Navigation oder Module, die eine Neos Webseite bilden, sind in einer Art Baum einsortiert. Dieser Baum-Graph nennt sich „TYPO3 Content Repository“. Die einzelnen Knoten, also Inhaltselemente, werden als „Nodes“ bezeichent. Jeder Node besitzt einen Namen und einen NodeType. Zusätzlich können noch Eigenschaften und sogenannte ChildNodes oder SubNodes, also Kindelemente für jeden Node definiert werden. TYPO3 Neos stellt einige vordefinierte NodeTypes zur Verfügung.

- TYPO3.Neos:Node, der Basistyp ohne weitere Eigenschaften
- TYPO3.Neos:Document, Inhaltselemente dieses Typs besitzen eine eigene URL
- TYPO3.Neos:Page, erbt von Document
- TYPO3.Neos:ContentCollection, alle Inhalte, die Teil einer Seite sind und keine eigene URL besitzen und weitere Kindelemente beinhalten
- TYPO3.Neos:Content, der Typ von einfachen Inhaltselementen und am häufigsten Erweiterte

NodeTypes können von anderen NodeTypes erben und somit dessen Eigenschaften übernehmen.

Zur Veranschaulichung folgt eine beispielhafte Darstellung eines TYPO3 Content Repository einer Unterseite.

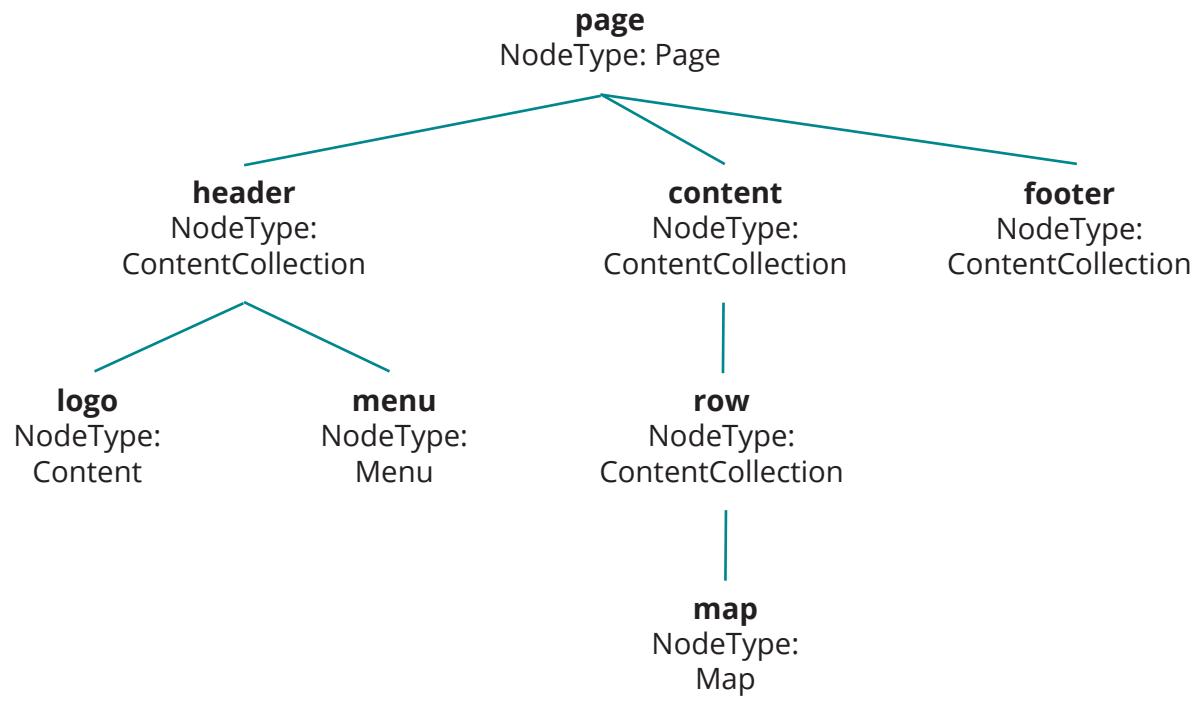


Abbildung 7.3: Beispiel einesTYPO3 Content Repository

Für ein individuelles Template werden viele neue NodeTypes angelegt oder bestehende angepasst. Für einen NodeType werden drei Dateien benötigt.

1. Die Datei *NodeTypes.yaml*, in der die neuen NodeTypes definiert und die bestehenden bei Bedarf erweitert werden. Der Name des Typs wird festgelegt, ebenso wie die Eigenschaften, die Eltern- und die Kindtypen.
2. Eine TypoScript-Datei (**.ts2*), welche die Schnittstelle zwischen Backend, also Dateneingabe und der Template-Datei bildet.

3. Eine Template-Datei (**.html*). Hier werden die Informationen, die von der TypoScript-Datei übergeben werden, durch die Template-Engine Fluid an die passende Stelle in das Markup gesetzt. (Vergleichbar mit der Template-Engine Handlebars, die in der Frontend-Entwicklung zum Einsatz kam.)

Begonnen wird mit dem Wurzel-Knoten des Baums. Dieser Knoten ist vom NodeType „Page“ und wird zuerst in der packageeigenen Datei *NodeTypes.yaml* erweitert.

```
 3
 4 'TYPO3.Neos.NodeTypes:Page':
 5   childNodes:
 6     'main':
 7       type: 'TYPO3.Neos:ContentCollection'
 8       constraints:
 9         nodeTypes:
10           'feneos.raschmidt:OneModuleRow': TRUE
11           'feneos.raschmidt:TwoModuleRow': TRUE
12           '*': FALSE
13     'footer':
14       type: 'TYPO3.Neos:ContentCollection'
15
```

Quellcode 7.1: *NodeTypes.yaml*, Definition des Typs *Page*

4 Der von Neos vordefinierte NodeType *Page* wurde bereits an einer anderen Stelle initial angelegt und wird hier nur um templatespezifische Eigenschaften erweitert.

5 Unter *childNodes* werden Nodes definiert, die beim Anlegen des NodeTypes, diesem automatisch hinzugefügt werden.

8, 15 Hier werden Bezeichner für die folgenden Kindelemente angegeben. Jeweils darunter wird

dem Kind ein Typ zugewiesen. Da diese ChildNodes strukturellen Zwecken dienen, bekommen sie den Typ *ContentCollection*.

10 Unter *contraints* stehen NodeTypes, die in diesem ChildNode erlaubt bzw. nicht erlaubt sind. In diesem Fall dürfen dem Kind *main* selbst nur Elemente der templateeigenen NodeTypes *OneModuleRow* und *TwoModuleRow* hinzugefügt werden. Alle anderen NodeTypes werden über Zeile 12 untersagt. Dabei ist die Reihenfolge egal, da die jeweils spezifischere Angabe berücksichtigt wird.

Nach jeder Änderung der *NodeTypes.yaml* muss in der Konsole der Befehl

```
flow.bat node:repair
```

auf den Projekt-Ordner ausgeführt werden.

Bei der Installation wurde bereits eine TypoScript-Datei für den NodeType *Page* angelegt. Die Datei der Wurzel nennt sich passenderweise *root.ts2*.

```
 1
 2  // Include TypoScript
 3 include: Prototypes/*ts2
 4
 5 page = Page {
 6   head {
       …
20   }
21
22   body {
23     templatePath = 'resource://feneos.raschmidt/Private/Templates/Page/
        Default.html'
24     sectionName = 'body'
25     parts {
26       mainMenu = Menu
27     }
```

```
28      content {
29        // Main content
30        main = PrimaryContent {
31          nodePath = 'main'
32        }
33        // Footer section
34        footer = ContentCollection {
35          nodePath = 'footer'
36        }
37      }
38      javascripts.site = TYPO3.TypoScript:Template {
39        templatePath = 'resource://feneos.raschmidt/Private/Templates/
Page/Default.html'
40        sectionName = 'bodyScripts'
41      }
42    }
43 }
```

Quellcode 7.2: *root.ts2*, Anlegen einer Instanz Page des Typs *Page*

3 Alle weiteren neuen oder erweiterten NodeTypes bekommen eine *.ts2*-Datei im Ordner *Prototypes*. Diese werden in die *root.ts2* inkludiert.

5 Eine Instanz namens „page“ des Typs *TYPO3.Neos.NodeTypes:Page* wird angelegt. Der Pfad „TYPO3.Neos.NodeTypes:“ gilt als Default und wird daher bei Typen-Angaben nicht weiter notiert.

6, 22 Ein NodeType kann beliebig tief geschachtelt werden, wobei die einzelnen Stufen Abschnitte, NodeTypes oder Variablen sein können. Die Instanz *page* wird in *head* und *body* unterteilt, equvialent zu den *head*- und *body*-Tags einer HTML-Seite.

23 Die Variable „templatePath“ gibt das Template der jeweiligen Instanz an. Für jeden Neos NodeType existiert bereits ein Template, dies kann jedoch durch ein eigenes Template, wie in diesem Fall, ersetzt werden.

31, 35 Der NodePath bezieht sich auf die in der *NodeTypes.yaml* definierten Bezeichner der ChildNodes.

24, 40 Die Variable „sectionName“ wird für Abschnitte einer NodeType-Instanz verwendet, die keinem Bereich durch die *NodeTypes.yaml* zugeordnet werden können und im HTML-Template angesprochen werden.

Die dritte Datei ist wie die *templates/layouts/default.hbs* des Frontend-Projektes. Sie bestimmt das Grundgerüst jeder Seite. Sie befindet sich im *Private*-Ordner des Package unter *Templates/Page* und nennt sich *Default.html*.

```
<!DOCTYPE html>
{namespace neos=TYPO3\Neos\ViewHelpers}
{namespace ts=TYPO3\TypoScript\ViewHelpers}
<html>
  <head>
    ...
  </head>
  <body>
    <f:section name="body">
      <header class="site-container">
        {parts.mainMenu -> f:format.raw()}
      </header>
      <div class="site-container" role="main">
        {content.main -> f:format.raw()}
      </div>
      <footer>
        {content.footer -> f:format.raw()}
      </footer>
    </f:section>
    <f:section name="bodyScripts">
      ...
    </f:section>
  </body>
</html>
```

Quellcode 7.3: *Default.html*, HTML-Grundgerüst der Webseite

1 Um ViewHelper (Funktionen, die das Rendern von Templates unterstützen) von Neos nutzen zu können, wird ein entsprechender Namespace angelegt.

6, 21 Hier werden Style- und Script-Dateien eingebunden. Im *head* wird noch ein Bereich für Meta-Tags angelegt, der in der *root.ts2* definiert wurde.

9 Der *body*-Bereich startet. Hier sind alle Variablen, Bereiche und Instanzen erreichbar, die in der *root.ts2* unter dem Pfad *page.body* angelegt wurden.

11, 14, 17 Ausdrücke der Template-Engine Fluid können auf zwei Arten geschrieben werden. Entweder Inline mit geschweiften Klammern wie in diesem Fall oder in einer Tag-basierten Syntax wie in den Zeilen 9 und 20. Die Funktion *format.raw()* sorgt dafür, dass der Inhalt der aufrufenden Instanz oder Variable so an die Stelle gesetzt wird, wie er ist. Das bedeutet, dass HTML-Tags, die in dieser Variable enthalten sind, mit ins Markup geschrieben werden, so dass der Browser diese interpretieren kann. Andernfalls würden die Tags als normaler Text im Browser angezeigt.

Wird nun das Backend neu geladen, steht als Ergebnis folgende Backend-Ansicht:

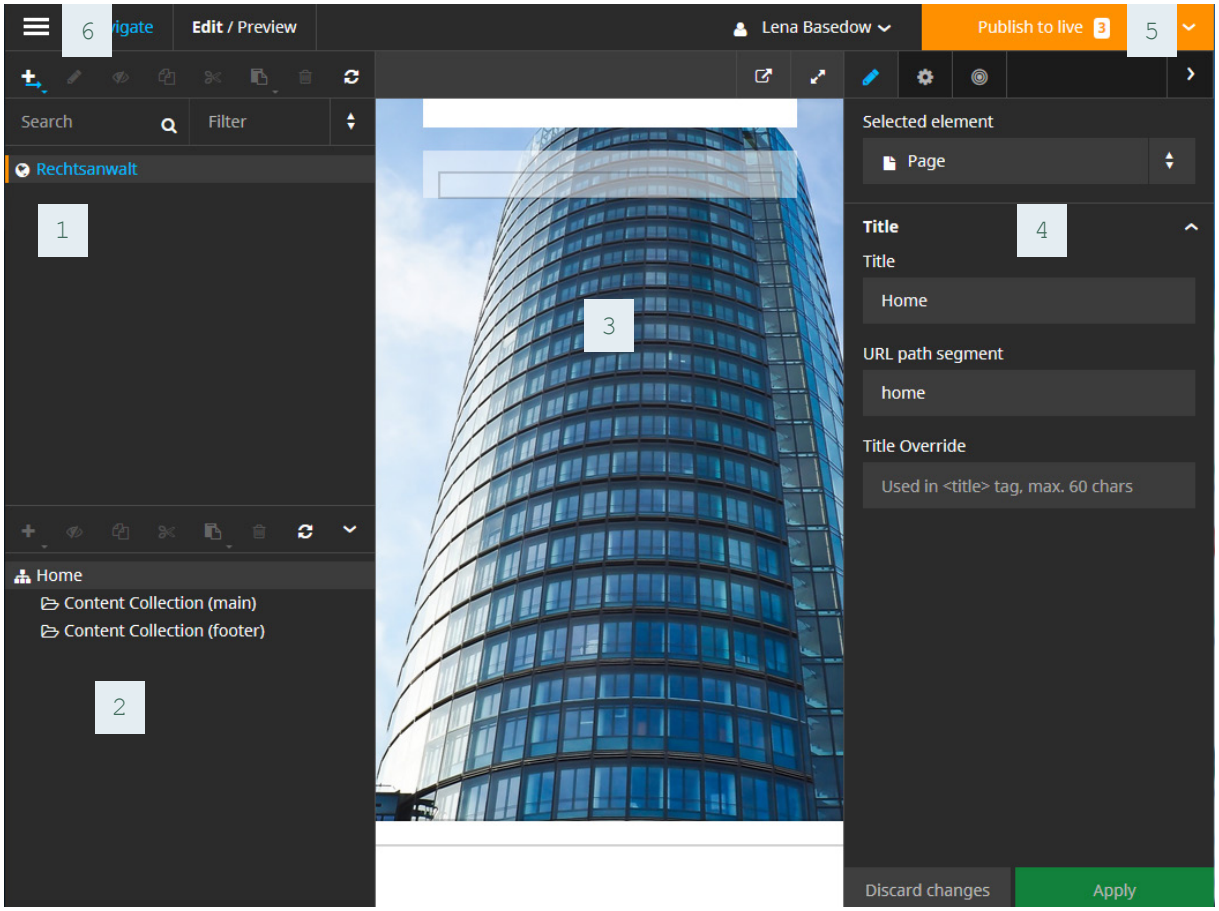


Abbildung 7.4: Aktuelle Ansicht des Neos Backends

1. Die Anzeige des Seitenbaumes
2. Die Inhaltselemente der unter 1. gewählten Seite
3. Die Webseite im Editiermodus
4. Der Inspector: Hier können NodeType-spezifische Einstellungen vorgenommen werden.
5. Über den „Publish to live“-Button werden Änderungen in der Liveseite veröffentlicht.
6. Das Backend-Menü beinhaltet neben der aktuellen Ansicht auch die Verwaltung von Medien, Usern, Plugins, Einstellungen, etc.

Nun werden die geplanten Unterseiten hinzugefügt.

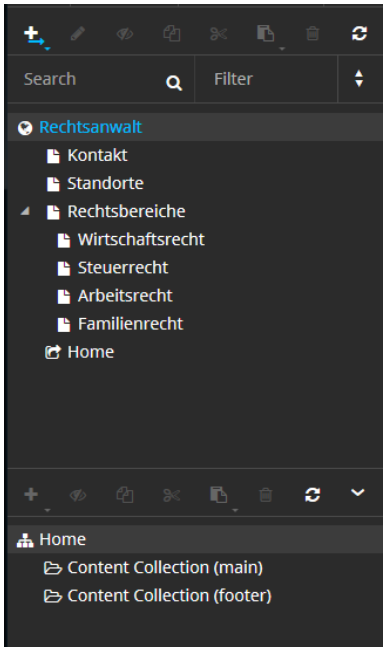


Abbildung 7.5: Ausschnitt des Backends: Seitenbaum mit neu angelegten Seiten

Durch den Klick auf das Plus-Symbol über dem Seitenbaum in Abbildung 7.5 (oben links) kann eine neue Seite angelegt werden. Der kleine Pfeil auf dem Plus bedeutet, dass neue Inhlate in das ausgewählte Element eingefügt werden. Durch langes Gedrückthalten der linken Maustaste lassen sich zusätzlich sowohl ein Pfeil nach oben sowie ein Pfeil nach unten auswählen, je nachdem was in der *NodeTypes.yaml* unter *constraints* definiert wurden. In diesem Fall ist allerdings der Pfeil nach innen möglich. Der Button öffnet ein neues Dialog-Fenster „Create new“ und stellt zwei Typen zur Auswahl: *Page*-Typ und *Shortcut*-Typ. Ein Shortcut ist eine Verlinkung auf eine interne oder externe Seite. Für die Unterseiten wird jeweils der Typ *Page* gewählt und für die Startseite *Rechtsanwalt* wird für die Generierung des Menüs ein Shortcut erstellt.

Für jede neue Seite des Typs *Page* wird, wie in der *NodeTypes.yaml* und in der *root.ts2* definiert, eine ContentCollection *main* und eine ContentCollection *footer* angelegt.

Header (Logo und Navigation)

Der Header besteht aus zwei Teilen, dem Logo und dem Menü. Für Beide werden entsprechende Einstellungen vorgenommen.

Die *NodeTypes.yaml* wird um einen Logo-Typen erweitert:

```
113
114 'feneos.raschmidt:Logo':
115   superTypes:
116     - 'TYPO3.Neos:Content'
117 # Content Elements
118
```

Quellcode 7.4: Ausschnitt aus *NodeTypes.yaml*

116 Dem Package *feneos.raschmidt* wird ein neuer NodeType namens „Logo“ hinzugefügt, der von *Content* erbt.

Als nächstes wird in der *root.ts2* der Pfad *page.body.parts* um ein Menu und ein Logo erweitert.

```
25      parts {
26        mainMenu = Menu
27        mainMenu {
28          templatePath = 'resource://feneos.raschmidt/Private/Templates/
Parts/MainMenu.html'
29          entryLevel = 1
30          maximumLevels = 2
31        }
32        logo = feneos.raschmidt:Logo
33        logo {
34          templatePath = 'resource://feneos.raschmidt/Private/Templates/
NodeTypes/Logo.html'
35        }
36      }
```

Quellcode 7.5: Ausschnitt der *root.ts2*, Erweiterung um ein Menü und ein Logo

26, 32 Zwei neue Instanzen werden angelegt. Ein „mainMenu“ von *TYPO3.Neos.NodeType:Menu* und ein Logo von dem eben in der *NodeTypes.yaml* definierten Typ „Logo“.

28, 34 Beiden wird ein eigenes Template zugewiesen.

29, 30 Hier werden das Einstiegslevel und das maximale Level an Untermenüstufen in die vordefinierten Menü-Variablen geschrieben. Das Wurzelverzeichnis der Website ist die Seite „Rechtsanwalt“ im Seitenbaum und wäre das Level 0. Damit diese trotzdem als Menüpunkt aufgenommen wird, wurde ein Shortcut erstellt.

Um die zwei neuen Elemente der *Page* auch im HTML-Template hinzuzufügen, wird die *Default.html* folgendermaßen erweitert:

```
23 <f:section name="body">
24    <header class="site-container">
25     {parts.logo -> f:format.raw()}
26     {parts.mainMenu -> f:format.raw()}
27    </header>
28    <div class="site-container" role="main">
```

Quellcode 7.6: Auschnitt *Default.html*, Ausbau des Headers

25, 26 An diesen Stellen wird der HTML-Code des Logos und des Menüs eingefügt.

Zu guter Letzt müssen noch die zwei Dateien mit dem entsprechenden Markup angelegt werden. Beide enthalten den HTML-Code, der während der Frontend-Produktion für das Logo und das Menü erstellt wurde. Statt Handlebars wird nun Fluid für das Einsetzen des Contents verwendet.

```
1
2 <a class="logo" href="\">
3   <img alt="Rechtsanwälte Schmidt" src="{f:uri.resource(path: 'img/
ralogo.png', package: 'feneos.raschmidt')}" />
4 </a>
5
```

Quellcode 7.7: *Logo.html*

3 Über den Fluid-ViewHelper *uri.resource* wird das Logo eingebunden. Der ViewHelper zeigt auf das *Public*-Verzeichnis des angegebenen Package.

```
 1 {namespace neos=TYPO3\Neos\ViewHelpers}
 2
 3 <nav class="nav-collapse clearfix">
 4   <span class="icon-menu icon-medium"></span>
 5   <ul class="clearfix first-level">
 6   <f:for each="{items}" as="item">  <!-- items kommt vom Prototypen Menu
und enhält alle Seiten bis zur angegebenen Grenze -->
 7     <f:if condition="{item.subItems}">
 8       <f:then>
 9         <li class="{f:if(condition:'{item.state} == "active"',
then:'active')} navigation-btn navigation-btn-first-level has-subnav">
10           <neos:link.node node="{item.node}" class="navigation-btn-first-
level-link">{item.label}</neos:link.node>
11           <ul class="second-level">
12             <f:for each="{item.subItems}" as="subItem">
13               <li class="navigation-btn navigation-btn-second-level">
14                 <neos:link.node node="{subItem.node}" class="navigation-
btn-second-level-link" />
15               </li>
16             </f:for>
17           </ul>
18         </li>
19       </f:then>
20       <f:else>
21         <li class="{f:if(condition:'{item.state} == "active"',
then:'active')} navigation-btn navigation-btn-first-level">
22           <neos:link.node node="{item.node}" class="navigation-btn-first-
level-link">{item.label}</neos:link.node>
23         </li>
24       </f:else>
25     </f:if>
26   </f:for>
27   </ul>
28 </nav>
```

Quellcode 7.8: *MainMenu.html*, Template des Hauptmenüs

6 Instanzen des *Menu*-NodeTypes enthalten automatisch eine Collection der zu rendernden Seiten, die unter „items“ gespeichert sind und hier über einen *for*-ViewHelper einzeln ausgelesen werden.

7–25 Jedes Item wird auf Unterseiten geprüft. Enhält es welche, wird das Markup im *f:then*-Teil ausgeführt, andernfalls wird der *f:else*-Teil gerendert.

9–21 Für jedes *li* des Hauptmenüs wird der aktuelle Status des Items geprüft und falls dessen Seite die aktuell angezeigte ist, bekommt es die CSS-Klasse „active“.

10, 22 Der Neos-ViewHelper „link.node“ erstellt einen Link zu dem unter „node“ angegebenen Knoten des Seitenbaums.

Im Header werden nun das Logo und die Punkte des Seitenbaums als Menü angezeigt (Abbildung 7.6).

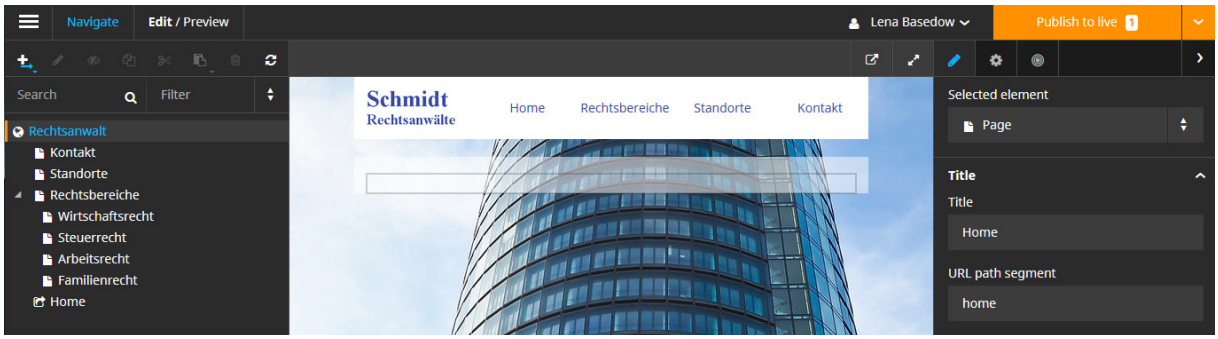


Abbildung 7.6: Backend-Ansicht mit Logo und Menü

OneModuleRow

Um die Seiten weiter auszubauen, müssen der Content-Collection *main* Reihen für die Module hinzugefügt werden. Die NodeTypes, die in der *NodeTypes.yaml*-Datei schon erwähnt wurden, werden als nächstes angelegt. Da der Ablauf der beiden Typen sehr ähnlich ist, wird hier nur das Anlegen der Reihe für ein Modul erklärt. Die *OneModuleRow* hat die Option auf einen weißen Hintergrund sowie eine Überschrift und kann keine weiteren Reihen enthalten. Diese Eigenschaften werden in der *NodeTypes.yaml* definiert (Quellcode 7.9).

```
19
20 'feneos.raschmidt:OneModuleRow':
21   superTypes:
22     - 'TYPO3.Neos:Content'
23   constraints:
24     nodeTypes:
25       '*': FALSE
26   childNodes:
27     'oneModuleRowContent':
28       type: 'TYPO3.Neos:ContentCollection'
29       constraints:
30         nodeTypes:
31           'feneos.raschmidt:OneModuleRow': FALSE
32           'feneos.raschmidt:TwoModuleRow': FALSE
33           '*': TRUE
34   ui:
35     label: 'Reihe für ein Modul'
36     group: 'raschmidt'
37     icon: 'icon-columns'
38     inlineEditable: TRUE
39     inspector:
40       groups:
41         rowsettings:
42           label: 'Einstellungen Reihe'
43   properties:
44     setBackground:
45       type: boolean
46       defaultValue: TRUE
47       ui:
48         label: 'Mit weißem Hintergrund?'
49         reloadIfChanged: TRUE
50         inspector:
51           group: 'rowsettings'
52     setHeadline:
53       type: string
54       ui:
55         label: 'Überschrift (optional)'
56         reloadPageIfChanged: TRUE
57         inspector:
58           group: 'rowsettings'
59
```

Quellcode 7.9: Ausschnitt *NodeType.yaml, Definition eines neuen NodeTypes*

23–25 Diesem Typ dürfen keine weiteren Kindelemente hinzugefügt werden, außgenommen davon sind die ChildNodes (Zeile 26).

26–33 Beim Hinzufügen einer *OneModuleRow* in eine Seite wird dieser automatisch ein Kindelement des Typs *ContentCollection* hinzugefügt, welches selber alles außer Reihen-Typen enthalten darf. Zu einem späteren Zeitpunkt werden noch weitere NodeTypes an dieser Stelle auf *FALSE* gesetzt.

34–42 Die Einstellungen für das Backend werden unter „ui“ vorgenommen. Hier wird als erstes der für den User lesbaren Namen („label“) sowie die Gruppe, unter der die Reihe beim „Create New“-Dialog aufgelistet wird, festgelegt. Zusätzlich werden diejenigen Gruppen angegeben, unter denen die nachfolgenden Eigenschaften im Inspector angezeigt werden sollen.

43 Die für diesen Typ gesetzten Eigenschaften, werden unter „properties“ aufgeführt.

44–51 Die erste Eigenschaft ist eine Variable, „setBackground“, vom Typ *boolean*. Hierdurch kann der User den Hintergrund über eine Checkbox im Inspector unter der Gruppe „rowsettings“, die in Zeile 41 definiert wurde, aktivieren oder deaktivieren. Das Label aus Zeile 48 wird neben der Checkbox stehen. Ändert sich der Wert der Checkbox wird das Element, dank Zeile 49, neu geladen.

52–58 Die zweite Eigenschaft, „setHeadline“, lässt den User in einem Textfeld, dank Zeile 53, im Inspector unter der Gruppe *rowssettings* eine Überschrift für diese Reihe eingeben.

Für ein besseres Verständnis werden in der folgenden Abbildung noch einmal die Zeilennummern, die das dadurch jeweils markierte Element ausgelöst haben, angezeigt. Der Screenshot zeigt das Neos Backend. In die ContentCollection *main* wurde ein Inhaltselement des Typs *OneModuleRow* eingefügt (blauer Rahmen in Abb. 5.8).

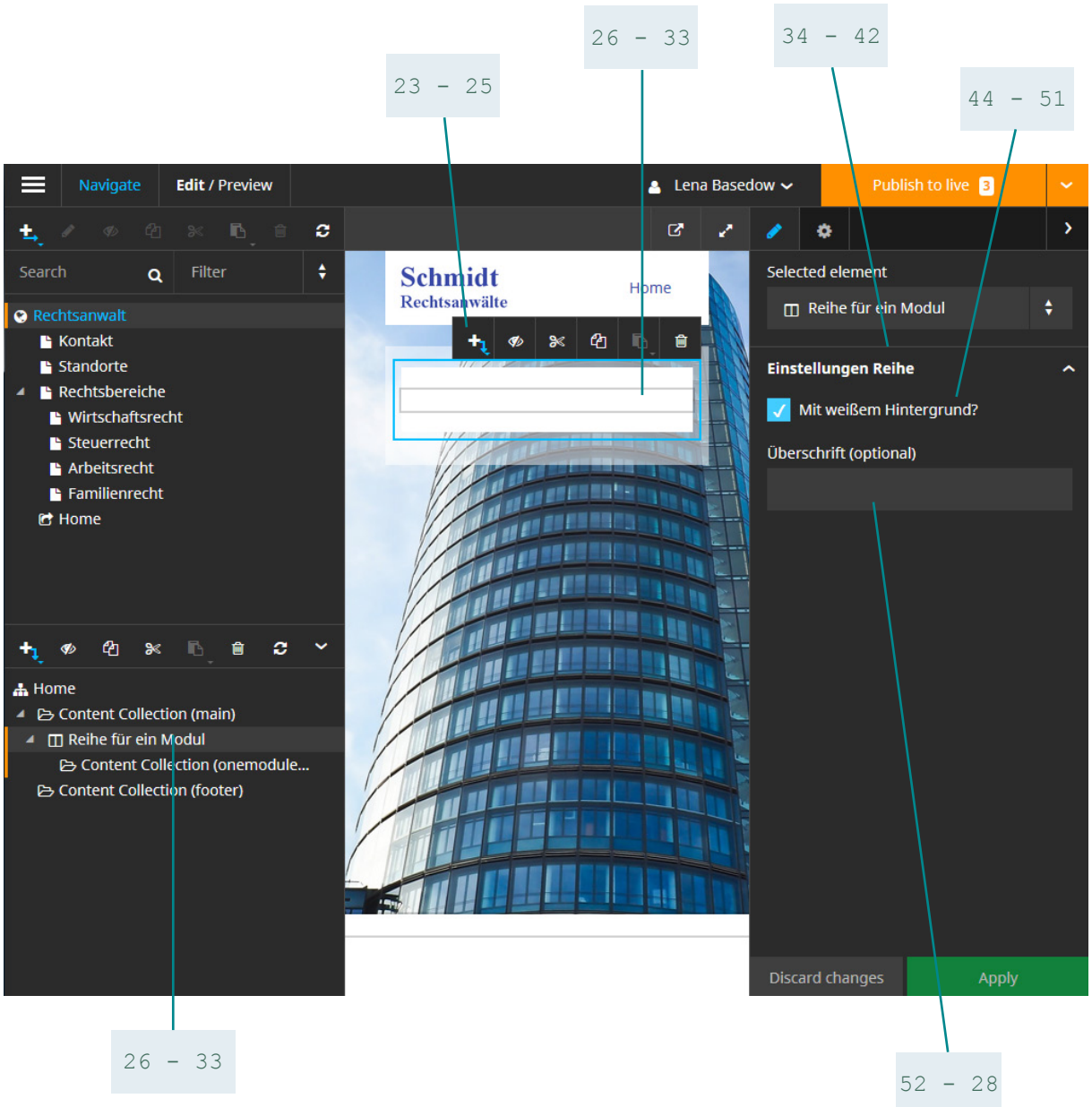


Abbildung 7.7: Seite mit einer *OneModuleRow*

Im Gegensatz zu PHP arbeitet TypoScript nicht mit Klassen sondern mit Prototypen. Für den neuen NodeType wird in einer eigenen TypoScript-Datei ein neuer Prototyp angelegt.

```
1 prototype(feneos.raschmidt:OneModuleRow) < prototype(TYPO3.Neos:Content)
2 {
3   templatePath = 'resource://feneos.raschmidt/Private/Templates/
NodeTypes/OneModuleRow.html'
4   oneModuleRowContent = TYPO3.Neos:ContentCollection {
5     nodePath = 'oneModuleRowContent'
6   }
7   setBackground = ${q(node).property('setBackground')}
8   headline = ${q(node).property('setHeadline')}
9 }
```

Quellcode 7.10: *OneModuleRow.ts2,* Definition des *OneModuleRow*-Prototypen

1 Der neue Prototyp "feneos.raschmidt:OneModuleRow" wird angelegt und erbt vom Neos NodeType *Content*.

4 Der Bereich des ChildNodes wird als ContentCollection deklariert.

7, 8 Über sogenannte FlowQuery-Ausdrücke (*{q()}*) werden die Werte der Variablen aus dem Backend an TypoScript-Variablen übergeben. Diese zwei neuen Variablen können in dem *OneModuleRow*-Template ausgelesen werden.

Das Markup für die OneModuleRow wird aus dem Frontend-Projekt in die neue Datei *OneModuleRow.html* kopiert und um Fluid-Audrücke ergänzt.

```
 2 <div class="one-module-row {f:if(condition: '{setBackground}', then:
'colored-bg')}">
 3   <f:if condition="{headline}">
 4     <f:then>
 5       <h2 class="h2">{headline}</h2>
 6     </f:then>
 7   </f:if>
 8   <div class="module">
 9     {oneModuleRowContent -> f:format.raw()}
10   </div>
11 </div>
```

Quellcode 7.11: *OneModuleRow.html*, Template des *OneModuleRow*-NodeTypes

2 Die Funktion *f:if* prüft den Wert der TypoScript-Variablen *setBackground*. Enthält sie den Wert *TRUE*, ist also die Checkbox im Inspector aktiviert, bekommt der *div*-Container zusätzlich die CSS-Klasse „colored-bg“.

3–7 Diesmal in Tag-basierter Syntax geschrieben, prüft eine *if*-Funktion, ob die Variable *headline* Text enthält. Ist dies der Fall, wird der Variablen-Inhalt als *h2*-Überschrift hinzugefügt.

9 Jedes Kindelement, welches der OneModuleRow bzw. dessen ChildNode hinzugefügt wird, wird durch die *format.raw()*-Funktion an dieser Stelle platziert.

Map

Das Karten-Modul soll ebenfalls Teil der Website werden. Die Herausforderung hierbei besteht in dem Einbinden der Marker in die Karte. Die Positionen der Marker werden über Koordinaten mit Hilfe von JavaScript bestimmt. Bei bisherigen Modulintegrationen wurden die Informationen vom Backend nur in HTML-Dateien eingebunden. Es wird eine Möglichkeit gesucht, wie der Kunde die Adressinformationen und Koordinaten mehrerer Standorte bequem im Backend eingeben kann und diese dann im JavaScript-Code verwendet werden, um die Standorte als Marker in der Karte anzeigen zu lassen.

Die Idee ist, dass der Kunde einen NodeType „Map“ der Webseite hinzufügt, der die Karte und einen Platzhalter für die Standortinformationen enthält. Für jeden Standort fügt er in den Platzhalter einen NodeType names „IconCardModulMap“ ein. Direkt auf der Webseite trägt er die Standortinformationen ein und in drei Eingabefeldern im Inspector die Koordinaten und den Standortnamen.

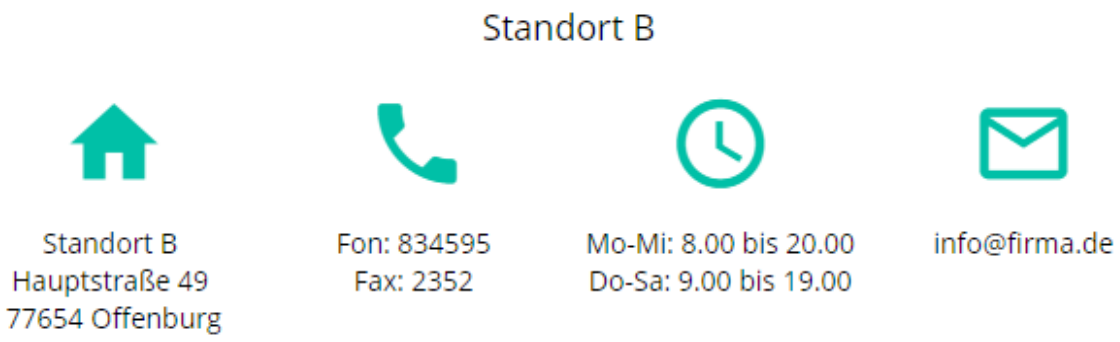


Abbildung 7.8: NodeType *IconModuleCardMap* im Frontend

Begonnen wird wie zuvor mit dem Definieren der neuen NodeTypes in der *NodeTypes.yaml*-Datei.

```
346 'feneos.raschmidt:IconModuleCardMap':
…
357   properties:
…
365     lat:
366       type: string
367       ui:
368         label: 'Breitengrad (latitude)'
369         help:
370           message: 'Die Koordinaten lassen sich über die Adresse auf
http://www.latlong.net ermitteln.'
371         inspector:
372           group: 'location'
373           editorOptions:
374             required: TRUE
…
381     address:
382       type: string
383       defaultValue: 'Name, Straße und Ort hier eintragen'
384       ui:
385         inlineEditable: TRUE
```

Quellcode 7.12: Ausschnitt *NodesType.yaml*, Definition des neuen NodeTypes *IconModuleCardMap*

370 Für die Eigenschaft „lat“ wird ein Hilfstext angegeben. Im Inspector erscheint neben dessen Label ein kleines Fragezeichen-Icon, das bei einem Mouseover einen Tooltip mit der eingetragenen Hilfsnachricht anzeigt.

374 Diese Eigenschaft wird als Pflichtfeld deklariert.

381 Für die Eigenschaft „address“ wurde keine Inspector-Gruppe angegeben, das hat den Effekt, dass im Inspector kein Eingabefeld dafür angezeigt wird. Da dieses Feld direkt auf der Webseite editiert werden soll, ist ein Eingabefeld im Inspector nicht nötig.

Für den NodeType *Map* werden Startkoordinaten, das initale Zoomlevel und ein Buttontext als Eigenschaften definiert. Alle drei sind im Inspector anzugeben. Als automatisch hinzugefügter ChildNode wird eine ContentCollection l*ocationCards* notiert, die nur Kindelemente des Typs „IconModuleCard“ enthalten darf (c*onstraints*).

Im nächsten Schritt werden die neuen Prototypen in TypoScript-Dateien geschrieben.

```
 1 prototype(feneos.raschmidt:Map) < prototype(TYPO3.Neos:Content) {
 2   templatePath = 'resource://feneos.raschmidt/Private/Templates/
NodeTypes/Map.html'
 3   locationCards = TYPO3.Neos:ContentCollection {
 4     nodePath = 'locationCards'
 5     attributes.class = attributes.class + ' locationinfo-placeholder'
 6   }
 7
 8
 9   locations = ${q(node).find('locationCards').children()}
10 }
```

Quellcode 7.13: *Map.ts2*, Definition des *Map*-Prototypen

3 Der ChildNode *locationCards* wird als ContentCollection angelegt.

5 Der Prototyp *ContentCollection* besitzt die Variable „attributes.class“. Eine ContentCollection wird beim Einfügen in das Markup in einen *div*-Container gesteckt, der als CSS-Klasse den Wert der Variablen *attributes.class* erhält. Hier wird dem bestehenden Wert noch die templatespezifische CSS-Klasse *location-placeholder* hinzugefügt.

9 Um die Marker in der Karte zu platzieren, werden dessen Koordinaten benötigt. Dazu werden

alle Kindelemente („.children()“) des eigenen Kindelementes („.find('locationCards')“) des aktuellen Knotens („q(node)“) in die Variable *locations* gespeichert. In dieser Variablen befinden sich demnach alle bestehende Inhaltselemente des NodeTypes *IconModuleCardMap*, die in die ContentCollection *locationCards* (Zeile 3) eingefügt wurden.

In der *IconModuleCardMap.ts2* werden alle für den NodeType definierten Eigenschaften an TypoScript-Variablen übergeben, um sie in der Template-Datei *IconModuleCardMap.html* verwenden zu können:

```
 1 {namespace neos=TYPO3\Neos\ViewHelpers}
 2 <div id=”{id}” class=”locationinfo”>
 3 {neos:contentElement.editable(property: 'name', tag:'h3', class:'h3')}
 4 <div class=”module col”>
 5   <div class=”single-icon-card”>
 …
10   </div>
11   <div class=”single-icon-card”>
…
14   </div>
15   <div class=”single-icon-card”>
…
18   </div>
19   <div class=”single-icon-card”>
…
22   </div>
23 </div>
24 </div>
```

Quellcode 7.14: Template-Datei für die Standortinformationen

2 Die CSS-Klasse *locationinfo* hat die Eigenschaft „display: none“, da die Information erst angezeigt werden soll, wenn auf den jeweiligen Marker geklickt wurde. Da das Inlineditieren nicht möglich ist, wenn das zu bearbeitende Element nicht angezeigt wird, muss der CSS-Code angepasst

werden. Neos fügt dem *body*-Tag im Backend-Modus die Klasse „neos-backend“ hinzu. Durch die zusätzliche Zeile im CSS: „.neos-backend .locationinfo {display: block;}“ lassen sich die Standortinformationen bearbeiten.

3 Über den Neos-ViewHelper *contentElement.editable* werden Variablen eingefügt, die direkt in der Seite editierbar sein sollen. Der Funktion werden drei Eigenschaften übergeben: *property* enthält die Variable mit dem Inhalt, der ausgegeben werden soll, *tag* enthält den HTML-Tag in dem der Inhalt gerendert werden soll und *class* enthält die CSS-Klasse, die dem HTML-Tag vergeben werden soll.

... Anstelle der Pünktchen werden die Standortinformationen eingefügt, die Adresse, die Kontaktnummern, die Öffnungszeiten und die E-Mail-Adresse.

```
 1 <div id="map"></div>
 2 <div class="clearfix map-button">
 3   <input type="button" class="btn pull-right" id="showAllMarkers"
value="{button}" />
 4 </div>
 5 {locationCards -> f:format.raw()}
 6
 7 <script type="text/javascript">
…
64 </script>
65
```

Quellcode 7.15: *Map.html*, Karten-Template mit dem JavaScript-Teil für die Karte und die Marker

1–4 Das aus der Frontend-Produktion bekannte Markup für das Karten-Modul wird eingefügt.

7 Um die Variablen aus der TypoScript-Datei im JavaScript-Code nutzen zu können, wird dieser in einem *script*-Tag direkt in das Template geschrieben.

Um den JavaScript-Code im Template nutzen zu können, wird der Ausdruck *<![CDATA[...]]>* verwendet. In den *script*-Tag kommt der Code, der während der Frontend-Produktion in der *function.js* unter *location* entstanden ist.

```
 7 <script type="text/javascript">
 8   <![CDATA[
 9     function initMap() {
…
25     };
26   ]]>
27   <![CDATA[
28     function setMarkers(map) { ]]>
29       <f:for each="{locations}" as="location">
30         <![CDATA[
31         var lat = ]]>{location.properties.lat}<![CDATA[ ;
32         var long = ]]>{location.properties.long}<![CDATA[ ;
33         var name = '']]>{location.properties.name}<![CDATA[' ;
34         var marker = L.marker([lat,long],{title: name}).addTo(map);
35         marker.name ='']]>{location.identifier}<![CDATA[' ;
36         marker.on('click', function() {
37           showInfo(this.name);
38         }); ]]>
39       </f:for>
40     <![CDATA[
41     };
42   ]]>
43   <![CDATA[
44     function showInfo(id) {
…
46     };
47   ]]>
48   <![CDATA[
49     function showAllMarkers() {
…
51     };
52   ]]>
53 </script>
```

Quellcode 7.16: Auschnitt *Map.html*, JavaScript-Teil zum Einbinden der Karte und der Marker

27-29 Die Trennung von JavaScript-Code und Fluid-Ausdrücken wird über den *CDATA*-Ausdruck erzeugt. Wobei jeder JS-Code davon eingeklammert wird. Außerhalb dieses speziellen Ausdruckes können TypoScript-Variablen dank Fluid eingefügt werden.

29 Durch die *for-each*-Schleife wird der Code in den Zeilen 30-38 für jedes Kindelement, also die „IconModulCardMap“s, das in der *locations*-Variable gespeichert wurde, ausgeführt.

31-33 Hier werden die Eigenschaften, die für jeden Node des Typs *IconModuleCardMap* im Inspector eingegeben wurden, zur Weiterverarbeitung durch JavaScript an entsprechende Variablen übergeben.

Alle weiteren Module werden nach dem selben Prinzip integriert und daher nicht weiter aufgeführt. Nach Beendigung der Intergration kann der Kunde seine Webseite verwalten und Inhalte intergrieren.

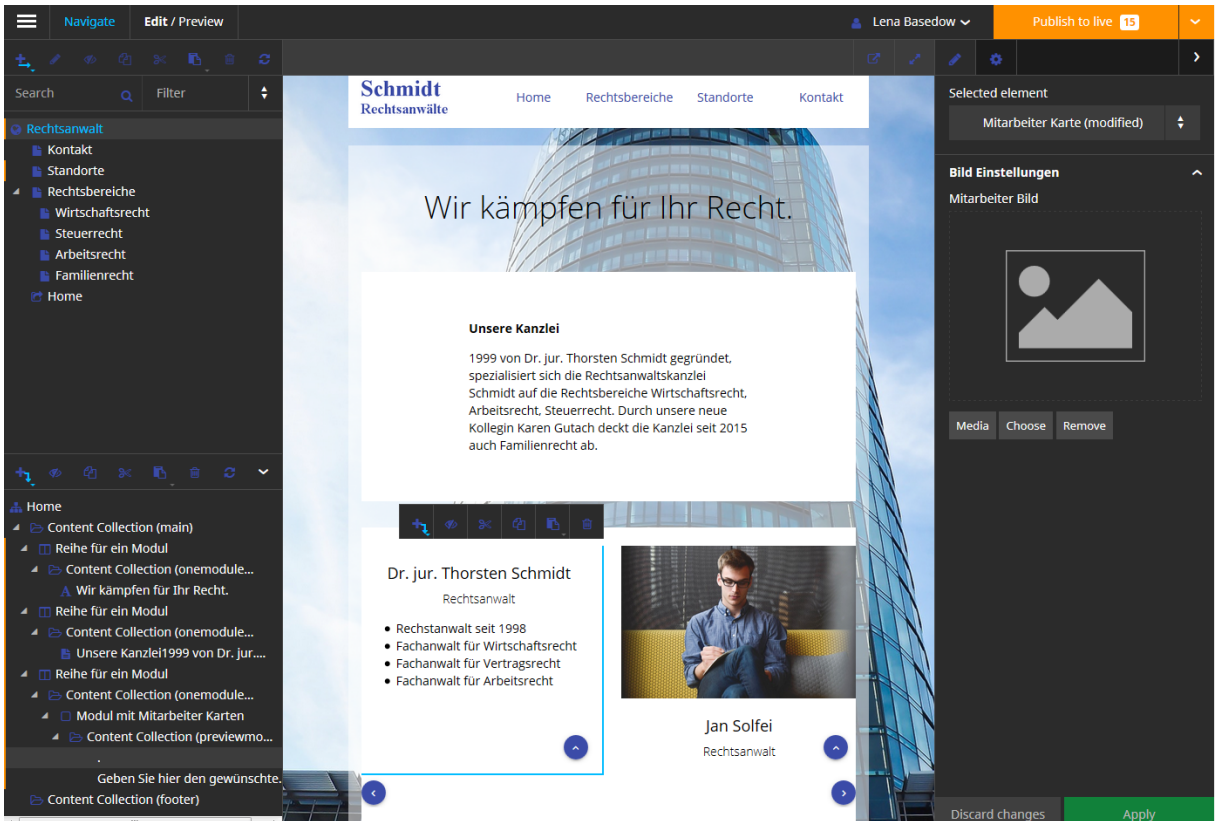


Abbildung 7.9: Ansicht der Startseite im Backend, wie sie vom Kunden bearbeitet wird.

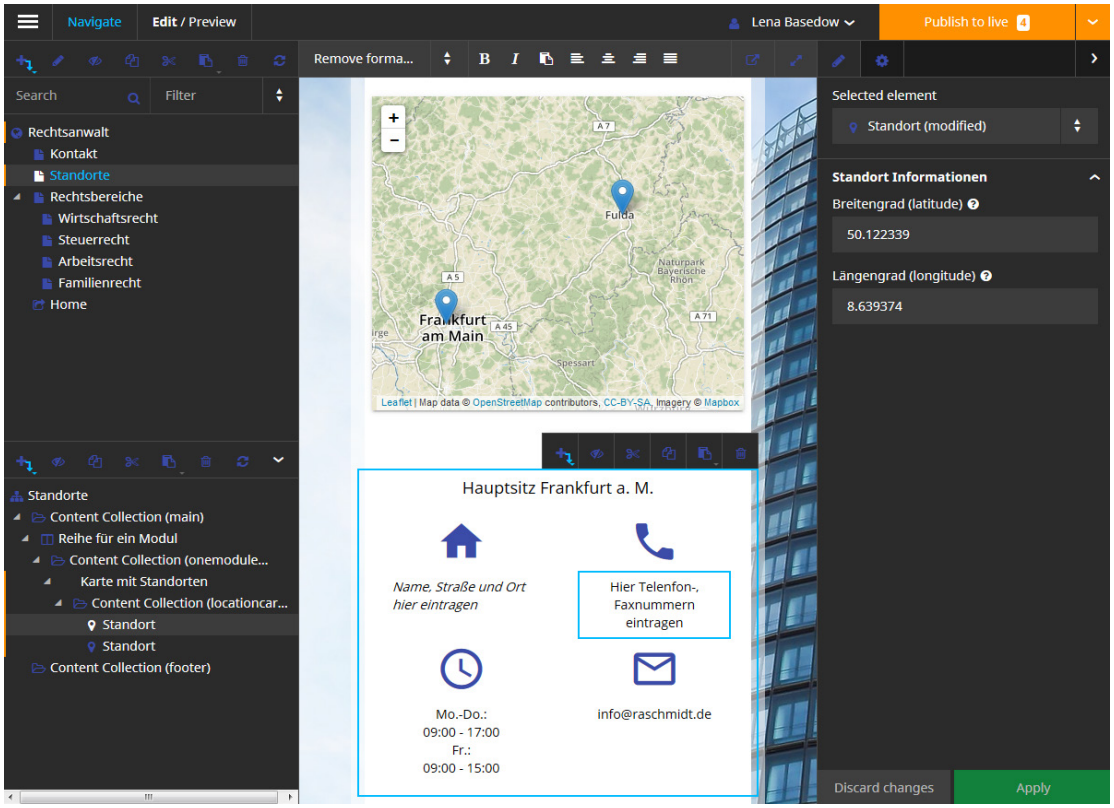


Abbildung 7.10: Ansicht der Seite *Standorte* im Backend wie sie vom Kunden bearbeitet wird.

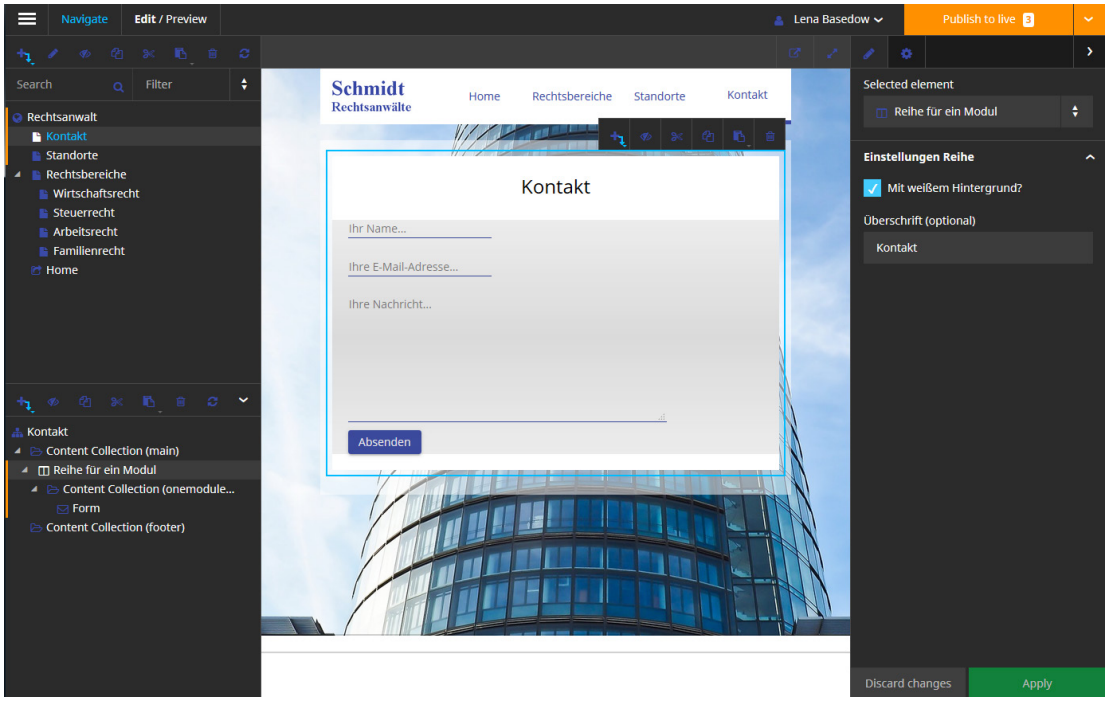


Abbildung 7.11: Ansicht der Seite *Kontakt* im Backend, wie sie vom Kunden bearbeitet wird.

8 Zusammenfassung des Workflows, Persönliches Fazit und Ausblick

Zusammenfassung des Workflows

In einer übersichtlichen Liste sollen nochmal alle Schritte des Workflows dargestellt werden:

1. Konzeptionsphase
   - 1.1 Grundlagen festlegen: ungefähre Zielgruppe, Module
   - 1.2 Erste Zeichnungen auf Papier
   - 1.3 Wireframes erstellen
   - 1.4 Design in Photoshop umsetzen
2. Entwicklung des Frontend-Templates
   - 2.1 Projekt einrichten
     - 2.1.1 Assemble-Projekt anlegen
     - 2.1.2 Ordnerstruktur anpassen
     - 2.1.3 erste Dateien anlegen
     - 2.1.4 Grunt konfigurieren
   - 2.2 Webdesign nach Atomic Design aufteilen
   - 2.3 Layout und Inhaltselemente technisch umsetzen
3. Integration in ein Content Management System
   - 3.1 Frontend-Template an Kundenwünsche anpassen
   - 3.2 CMS lokal installieren
   - 3.3 Projekt-Dateien („assets“) einbinden
   - 3.4 Layout und Module dynamisieren
   - 3.5 Inhalte einpflegen oder dem Kunden übergeben

Für zukünftige Projekte kann der Punkt 2 ausgelassen werden, da nun ein Grundordner mit dem eingerichteten Projekt existiert und nur noch kopiert werden muss. Ebenfalls kann der Schritt 1 und 2 ignoriert werden, wenn sich ein Kunde für ein bereits entwickeltes Frontend-Template entscheidet. In diesem Fall bedarf es nur noch der Integration in ein Content Management System.

Persönliches Fazit und Ausblick

Die Erstellung dieser Arbeit brachte mich als Webentwicklerin um einiges voran. Ich setzte mich intensiv mit neuen Techniken auseinander und erarbeitete einen Workflow, der modern und effektiv ist. Der Workflow deckt alle Phasen eines Webprojektes ab, von einem leeren Blatt Papier für Ideen bis hin zur Abnahme der Website mit CMS durch einen Kunden sowie abgeschlossene Schritte, die in weiteren Projekten aufgegriffen werden können.

In der Konzeption, der Frontend-Entwicklung sowie der Integration in das CMS TYPO3 Neos kamen mir bis dahin unbekannte Aspekte vor.

Die Frontend-Produktion mit einem Static Site Generator zeigte mir eine Möglichkeit, kleinere Web-Projekte ohne CMS zu realisieren. Allerdings gibt es bei einem SSG-Projekt viel größere Einstiegshürden als bei der Arbeit mit PHP. Die vielen Abhängigkeiten von einem Package Manager, einer Template-Engine oder einem Task Runner kosten unerfahrenen Entwicklern einige Zeit an Vorbereitung.

Ebenso das Arbeiten nach Atomic Design war neues Terrain für mich. Die Entwicklung des Frontends durchlief ich zweimal. Einmal ohne und einmal unter Berücksichtigung des Atomic Designs. Im zweiten Durchgang bemerkte ich, dass ich teilweise schon implizit nach diesem Prinzip entwickelt hatte, nichtsdestotrotz konnte ich noch einiges an CSS-Code einsparen, da ich an manchen Stellen denselben Code für ein Atom in zwei verschiedenen Modulen doppelt geschrieben hatte. Solche Redundanzen werden durch Atomic Design vermieden. Jedoch ergab sich dabei ein Overhead an Dateien, da teilweise jedes Atom, Molekül usw. in einzelne Dateien ausgelagert wurden. Der Zweck

von Atomic Design, die Übersichtlichkeit zu fördern, setzt sich dadurch selbst etwas außer Kraft.

Die größte Herausforderung dieser Arbeit war das Integrieren des Templates in das CMS TYPO3 Neos. Die Installation und Einarbeitung kostete aufgrund geringer Informationsressourcen im Internet einiges an Zeit. Doch als diese Phase erfolgreich überstanden war und ich das Prinzip des Aufbaus einer Neos Webseite verstanden hatte, lief die Integration schnell und mit Freude über die Hand.

Trotz aller Schwierigkeiten werde ich diesen Workflow beibehalten und TYPO3 Neos oder, je nach Größe der Seite, eine statische Webseite zukünftigen Kunden empfehlen.

Auch aufgrund stetig steigenden Interesses an Static Site Generatoren (Abbildung 9.1) werden diese wohl in Zukunft immer häufiger im Web Anwendung finden.

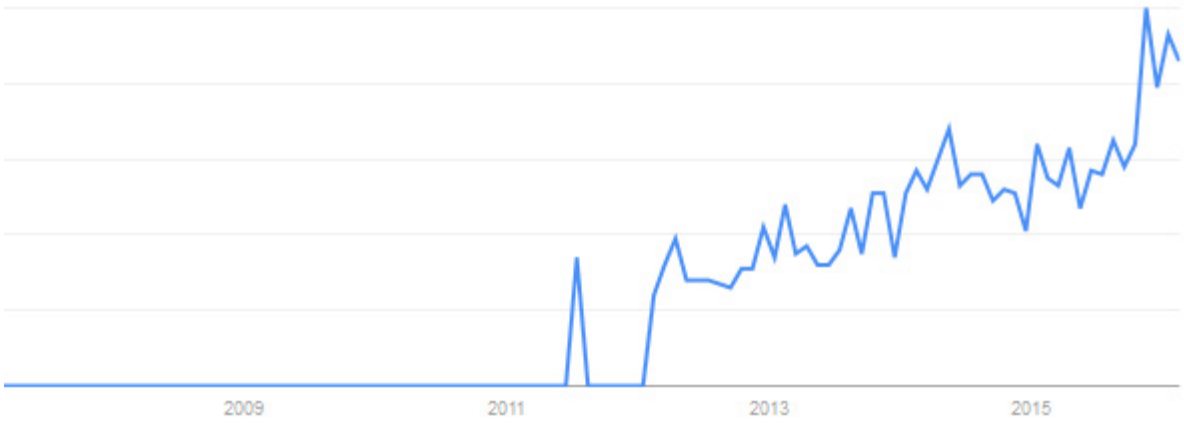


Abbildung 8.1: Zeitlicher Verlauf des Interesses an Static Site Generatoren (Google Trends, 02.2016)

9 Quellenverzeichnis

api.jquery.com Von http://api.jquery.com/ abgerufen am 02.01.2016

assemble.io [2015] Von http://assemble.io/ abgerufen am 30.12.2015

awesemthemes.de [12.2015] Von http://www.awesomethemes.de/wordpress-themes-kaufen/ abgerufen am 08.12.2015

Bauer, T. [01.2016] onlinemarketing.com. Von http://onlinemarketing.de/news/webdesign-2016-auf-diese-6-trends-sollten-marketer-sich-einstellen abgerufen am 02.02.2016

behance.net [12.2015] Von https://www.behance.net/gallery/8786901/FREE-Flat-Social-Icons-EPS abgerufen am 14.12.2015

bitballoon.com [2016] Von https://www.bitballoon.com/blog/2014/04/28/grunt-bitballoon abgerufen am 01.03.2016

Bonset, S. [07.2015] t3n.de. Von http://t3n.de/news/google-enthuellt-material-design-621169/ abgerufen am 14.12.2015

bradfrost.com [03.2012] Von http://bradfrost.com/blog/web/responsive-web-design-missing-the-point/ abgerufen am 08.12.2015

Brinkmann, F. [11.2015] t3n.de Von http://www.t3n.de/news/kostenlose-wordpress-themes-responsive-webdesign-376838/ aberufen am 08.12.2015

commons.wikimedia.org Von https://commons.wikimedia.org/wiki/File:LESS_Logo.svg abgerufen am 01.03.2016

design.google.com/icons [2015] Von https://design.google.com/icons/ abgerufen am 14.12.2015

docs.npmjs.com [2015] Von https://docs.npmjs.com/getting-started/what-is-npm abgerufen am 29.12.2015

dotcominfoway.com [08.2012] Von http://www.dotcominfoway.com/blog/responsive-web-design-infographic/ abgerufen am 08.12.2015

Droste, M, „Static Website Engines“, *Screenguide*, 01-03/2016, S.

elegantthemes.com [12.2015] Von http://www.elegantthemes.com/ aberufen am 08.12.2015

elmastudio.de [12.2015] Von http://www.elmastudio.de/wordpress-themes/faq aberufen am 08.12.2015

Ettisberger, A. [03.2015] youengineering.com. Von http://www.youengineering.com/blog/gruntjs-einstieg/ abgerufen am 30.12.2015

flatuicolors.com [12.2015] Von http://flatuicolors.com/ abgerufen am 14.12.2015

flavsrealism.com [02.2016] Von http://www.flatvsrealism.com/ abgerufen am 02.02.2016

Frost, B. [06.10.2013] bradfrost.com. Von http://bradfrost.com/blog/post/atomic-web-design/ abgerufen am 14.01.2016

getmdl.io [2015] Von http://www.getmdl.io/customize/index.html abgerufen am 14.12.2015

github.com [2016] Von https://github.com/assemble/generator-assemble a

google.com/design [2015] Von https://www.google.com/design/spec/style/typography.html#typography-typeface abgerufen am 14.12.2015

google.com/design [12.2015] Von https://www.google.com/design/spec/material-design/introduction.html abgerufen am 14.12.2015

google.com/design [12.2015] Von https://www.google.com/design/spec/what-is-material/environment.html abgerufen am 14.12.2015

google.com/design [12.2015] Von https://www.google.com/design/spec/what-is-material/material-properties.html abgerufen am 14.12.2015

google.com/design [12.2015] Von https://www.google.com/design/spec/what-is-material/elevation-shadows.html abgerufen am 14.12.2015

google.com/fonts [12.2015] Von https://www.google.com/fonts/specimen/Open+Sans abgerufen am 14.12.2015

google.de/fonts [2015] Von https://www.google.com/fonts abgerufen am 14.12.2015

Hahn, M. [2015]. Webdesign. „Das Handbuch zur Webgestaltung“ Bonn: Galileo Design

handlebarsjs.com [2015] Von http://handlebarsjs.com/ abgerufen am 30.12.2015

klickkomplizen.de [03.2010] Von http://klickkomplizen.de/blog/print-design/webdesign-print-design/professionelles-webdesign-oder-fertiges-webseiten-template/ abgerufen am 03.02.2016

Krahmer, E. [29.01.2016] netzsieger.de. Von https://www.netzsieger.de/ratgeber/wordpress-vs-joomla abgerufen am 29.02.2016

mittwald.de [2016]. Von https://www.mittwald.de/

mittwald.de [2016]. Von https://www.mittwald.de/neos abgerufen am 28.02.2016

mrknowing.com [25.07.2013] Von http://www.mrknowing.com/2013/07/25/eigenen-html-kalender-erstellen-html-und-javascript/ abgerufen am 08.01.2016

neos.readthedocs.org [01.2016] Von http://neos.readthedocs.org/en/stable/

netupdater.de [2016] Von http://www.netupdater.de/netupdater-de/content-management-system-cms/65/ abgerufen 23.02.2015

npmjs.com [2015] Von https://www.npmjs.com/ abgerufen am 29.12.2016

npmjs.com [2016] Von https://www.npmjs.com/package/grunt-resize-crop abgerufen am 03.01.2016

Schwarz, N., *Screen Guide,* 01-03/2016, S.3

t3n.de/tag [12.2015] Von http://t3n.de/tag/flat-design aberufen am 14.12.2015

t3n.de/tag [02.2016] Von http://t3n.de/tag/wordpress aberufen am 27.02.2015

t3premium.com/blog [12.2015] Von http://www.t3premium.de/blog/flat-design-vs-material-design/ abgerufen am 14.12.2015

Thattil, S. [08.06.2014], yuhiro.de. Von http://www.yuhiro.de/vor-und-nachteile-von-wordpress/ abgerufen am 28.02.2016

themeforest.net [12.2015] Von http://themeforest.net/?ref=awesem_DE abgerufen am 08.12.2015

templatemonster.com [12.2015] Von http://www.templatemonster.com/de/ abgerufen am 08.12.2015

typografie.info [2015} Von http://www.typografie.info/3/page/Schriften/fonts.html/_/open-sans-r438

Venkatesh, C.R. [08.2012] dotcominfoway.com. Von http://www.dotcominfoway.com/blog/responsive-web-design-infographic/ aberufen am 08.12.2015

w3techs.com Von http://w3techs.com/technologies/overview/content_management/all abgerufen am 26.02.2016

webkalkulator.com Von http://www.webkalkulator.com/cmsvergleich abgerufen am 26.02.2016

wikipedia.org Von https://de.wikipedia.org/wiki/Paketverwaltung abgerufen am 29.12.2015

wikipedia.org Von https://de.wikipedia.org/wiki/JQuery abgerufen am 02.01.2016

wikipedia.org Von https://en.wikipedia.org/wiki/File:JQuery_logo.svg abgerufen am 01.03.2016

wikipedia.org Von https://de.wikipedia.org/wiki/Joomla abgerufen am 28.02.2016

wpzoom.com [12.2015] Von http://www.wpzoom.com/themes/ abgerufen am 08.12.2015